WL-TR-95-1164

TFWGCON '95 TEST FACILITY WORKING GROUP CONFERENCE - 1995

PROCEEDINGS AND APPENDIX A OF PROCEEDINGS

CHARLES P. SATTERTHWAITE

JUNE 1995

FINAL REPORT FOR 06/12/95-06/16/95

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

NOTICE

When Government drawings, specifications or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patent invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Charles P. Satterthwaite
CHARLES P. SATTERTHWAITE
TFWGCON 95 Papers Chairman
WL/AAAF-2

James S. Williamson
JAMES S. WILLIAMSON, Acting Chief
Software Concepts Section
WL/AAAF-2

William Baker
WILLIAM R. BAKER, Acting Chief
Avionics Logistics Branch
WL/AAAF

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify WL/AAAF, WPAFB OH 45433-7301 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUN 1995	FINAL 06/12/95--06/16/95

4. TITLE AND SUBTITLE TFWGCON '95 TEST FACILITY WORKING GROUP CONFERENCE - 1995 PROCEEDINGS AND APPENDIX A OF PROCEEDINGS

5. FUNDING NUMBERS
C
PE 78611
PR 3090
TA 02
WU 27

6. AUTHOR(S) CHARLES P. SATTERTHWAITE

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-95-1164

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

THIS PROCEEDINGS AND ITS APPENDIX CONTAINS TECHNICAL PAPERS AND BRIEFINGS FROM THE 1995 TEST FACILITY WORKING GROUP CONFERENCE WHICH WAS HELD IN LAS VEGAS, NEVADA, 12-16 JUNE 1995.

14. SUBJECT TERMS
INSTRUMENTATION, EMBEDDED SOFTWARE, SOFTWARE METHODS AND PROCESSES, THREAT GENERATION, DATA ANALYSIS, DOCUMENTATION AND DESIGN

15. NUMBER OF PAGES 656

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TFWGCON '95

TEST FACILITY WORKING GROUP CONFERENCE

"The Avionics and Weapons System Software Development, Test, Support and Readiness Technology Forum Among Industry, Academia and Government"

12-16 June 1995

Las Vegas, Nevada

CONTENTS

INTEGRATED INTELLIGENT DIAGNOSTICS SUB-WORKING GROUP (I2DWG)

Co-Chairs:
John Camp and Charles Bosco

ATPS: An Expert Systems-Based Automated Test Planning System

Karen Okagaki*

Science Applications International Corporation
10260 Campus Point Drive
San Diego, California 92121
(619) 546-6515

Richard Ledesma/COL Chuck Cook

OUSD(A&T)/DTSE&E
The Pentagon, Room 3D1075
Washington, DC 20301

Abstract

The Automated Test Planning System (ATPS) is a rule-based expert system designed to aid OSD and Service staffs in their testing missions. ATPS provides the DTSE&E with an intelligent system for assessing program risk, harmonizing key acquisition documents, and building and reviewing Test and Evaluation Master Plans (TEMPs). The four primary components of the ATPS framework are TEMP Build, T&E Program Risk Assessment (TEPRAM), TEMP Review and T&E Program Design. This "system-of-systems" is being built sequentially and incrementally. Phase I was a feasibility study to determine the scope of ATPS and gather the information to be contained in the first module, TEMP Review. Phase II built this module, which was successfully fielded in 1993. Phase III saw the development of the TEPRAM, which was fielded in 1994. Phase IV developed the TEMP Build Module, released 1 March 1995. The final module, T&E Program Design, will be a management tool designed to provide oversight of the entire program, to provide summary information regarding the program's progress, and to provide management and other component staffs with the utmost flexibility in using and managing each ATPS module.

Introduction

As the Test and Evaluation (T&E) community continues to streamline the test planning and review process in the face of budget cutbacks and personnel consolidation, a need for productive tools for T&E analysts arises. Such tools should not replace the analyst, but should act as intelligent automation aids to provide insight and guidance in the test planning and review process. Expert systems-based tools often can provide the essential automation required, while targeting a wide range of potential end-users. Expert systems can benefit the most inexperienced test planner or analyst by bringing him up to speed quickly and efficiently, and by providing him with high-quality corporate knowledge. Even experienced analysts can benefit by the structure and consistency the software provides in the test planning and review process.

OUSD(A&T)/DTSE&E has supported the development of one such automation tool, ATPS, to help capture, maintain, and distribute corporate knowledge within the T&E community. ATPS is an expert systems-based software package that provides the user with information gleaned from composite knowledge gained from DoD testing organizations, extant paper checklists, and guidance derived from the DoD 5000 series instructions. The goal of ATPS is to improve the analyst's productivity by automating key components of the test planning mission.

Approach

A successful expert system requires that the domain be well-bounded, that subject matter experts be available, that using AI techniques adds value, and that there is an appropriate evaluation time to test the system in order to ensure that the knowledge it contains is complete, consistent, and useful.

The domain must be well-bounded. This isn't to imply that the problem be trivial, rather, it should be one where it requires some amount of expertise to perform well. In the larger "test planning and review" world, several smaller, more bounded domains are defined for inclusion into the ATPS concept: TEMP Build, TEPRAM, TEMP Review, and T&E Program Design. While these processes cannot be fully automated, key component capabilities within these domains are quite appropriate for automation.

Subject matter experts must be available. The T&E community is rich in written guidance and high-level, experienced test planners. In the development of ATPS, Service experts provide their input on every module. Quarterly Working Group meetings serve as a forum for analytical exchange between Service end-user representatives and the development team. Once per year, a Senior Advisory Group meets to review the past year's effort, and to guide the direction of new component development. The Senior Advisory Group consists of senior-level OSD and Service officials.

AI must add value. High turn-over in personnel has lead to a deficit in corporate knowledge and an inadequate transfer of that knowledge to new analysts and junior personnel. In spite of the apparent abundance of information, it is often a daunting task to try to coordinate the information in a useful and efficient manner. By identifying the key component areas of automation, and applying successful AI techniques, corporate knowledge is maintained and distributed in a highly effective manner. In the test planning and review domain, checklists or "to do" lists are a straight-forward means of implementing a consistent baseline for development and review of key acquisition documents. These checklists are implemented as forward-chaining rules that match on certain session parameters to determine which checklist questions to present to the analyst.

There must be an appropriate period of evaluation. Often in the past, AI and expert systems have been oversold, leading to disappointed and disillusioned users. It has always been the goal of ATPS to maintain the "analyst-in-the-loop," and to act as an aid to the analyst, not as a replacement for critical thought. Only a human analyst can make the correct critical judgments necessary in the test planning mission. In order to ensure that ATPS is providing the correct level of support to the human analyst, it is important to have the opportunity to field ATPS to a limited number of users, and to receive feedback for its continued improvement. This was accomplished in all phases of development, primarily through cooperation of the Working Group members.

During each development Phase, there has been a period of review and evaluation. In Phase I, a feasibility study was conducted to gather information and to identify potential end-user sites who would benefit from using ATPS. The Working Group was tasked to

choose an initial module for development based upon the fact-finding information. In Phase II, a prototype system, the TEMP Review Module, was developed and fielded. During this phase, the Working Group had time to evaluate the knowledge, "rules of thumb," and advisory information that would go into the system and make up the TEMP Review checklist. On completion of software development, approximately 20 Working Group members and OSD representatives beta tested the software and provided feedback for improvement. Upon fielding ATPS TEMP Review to the community at large, Working Group representatives were "trained as trainers" in the use of ATPS, and then went on to train others at their site. A mechanism for feedback was implemented so that users could receive support as needed.

In subsequent phases of development, new ATPS modules are beta tested by Working Group members first, and then released for general distribution as discussed above.

OUSD(A&T)/DTSE&E is committed in their support of ATPS as a means to provide the Services with a useful analysis tool that will improve the overall quality of Test and Evaluation planning. All Services remain fully aboard in their support of ATPS.

Architecture

We specify a modular ATPS design consisting of 4 major components as shown in Figure 1. The TEMP Build Module, is a set of Service specific checklists that aid in the development of the TEMP. The output of the TEMP Build module is a rough-draft TEMP in the 5000.2-M format. The T&E Program Risk Assessment Module (TEPRAM) helps the analyst harmonize key acquisition documents and provides a structure and process for assessing program risk. The TEMP Review Module, is the "final exam" for reviewing TEMPs. It contains a checklist of key points and guidance derived primarily from the DoD 5000 series documents. The T&E Program Design Module, is currently in the conceptual state. This module is envisioned as an intelligent management tool that will provide summary information on the program and allow access into any module within ATPS.

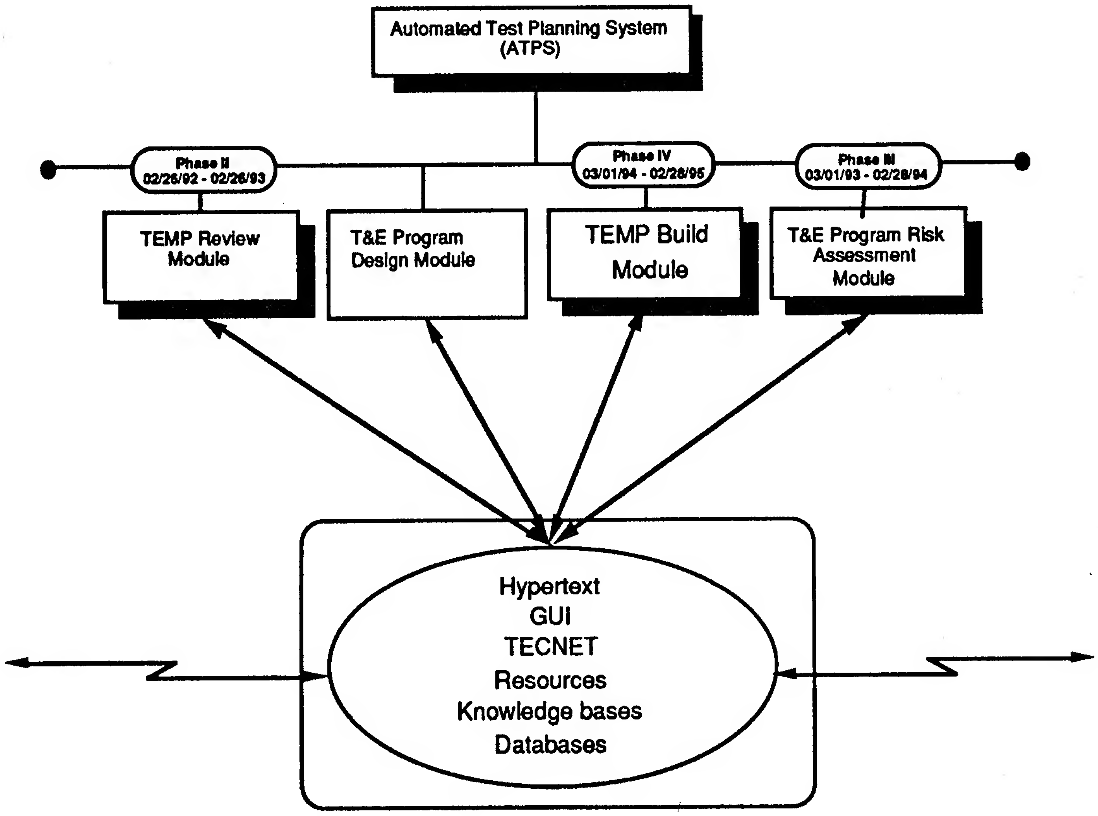


Figure 1

The ATPS software consists of a graphical user interface that seamlessly brings forward expert systems-based tools, hypertext help, word processing tools, and cross-reference capabilities that are common across the modules. Individual modules differ primarily in which rulebases are used to generate the checklists. This gives ATPS great extensibility. When a new module is added, the majority of the effort is in developing a new rule base. The user interface is changed only moderately to accommodate the new module.

Implementation

ATPS is hosted on a PC platform (486 PC) running Windows 3.11. It is developed in Microsoft C 7.0, the Rule-Extended Algorithmic Language (RAL), the Microsoft Windows Software Development Kit (Microsoft Windows SDK), and the Microsoft Multimedia Viewer Publishing Toolkit. A Macintosh version of ATPS is also supported. The Macintosh version is developed in MPW C and RAL for Macintosh.

An expert system attempts to mimic the way a human attempts to solve a particular problem. Humans tend to use "rules of thumb", shortcuts gained from experience, or follow a particular line-of-reasoning when solving a problem. Experienced analysts attempt certain test and evaluation tasks by following guidance provided to them in the form of outlines, formats, or checklists. The checklist questions are usually relatively straightforward, with some "yes/no" type branching which leads to a very straightforward expert systems implementation using forward chaining production rules. By linking checklist questions with additional knowledge (rules of thumb), guidance, and references gleaned from senior-level OSD and Service action officers , as well as from the DoD 5000 series documents, a very comprehensive and useful tool emerges for test and evaluation planning and review.

The ATPS rulebase consists of rules that have both condition and action parts. The condition, or left hand side of the rule, must be satisfied before the action, or right hand side of the rule, can occur. The rules differ from if-then statements in that the order of the rules is unimportant. It is the job of the run-time system to select which rule is applicable, given the current data in memory, and to execute its action. In ATPS, a sample rule represents a checklist question. Its condition part matches data elements in memory that may include Milestone, ACAT, and/or Service. Its action part includes displaying a checklist question, cross referencing the question to the DoD 5000.2-M, and linking the question to Advisor information. Other rules in ATPS enable the user to back up to previous questions, and clean up working memory between sessions.

The Common Framework

Figure 2 shows the main ATPS interface screen as it appears in the TEMP Build Module (ATPS Release 4.0). The main screen varies slightly between modules, however, certain core components are common throughout. The top of the screen shows the Intelligent Checklist area. This is where checklist questions are displayed to the user. At the bottom of the screen, the Comments Editor allows the user to type in responses to the checklist questions.

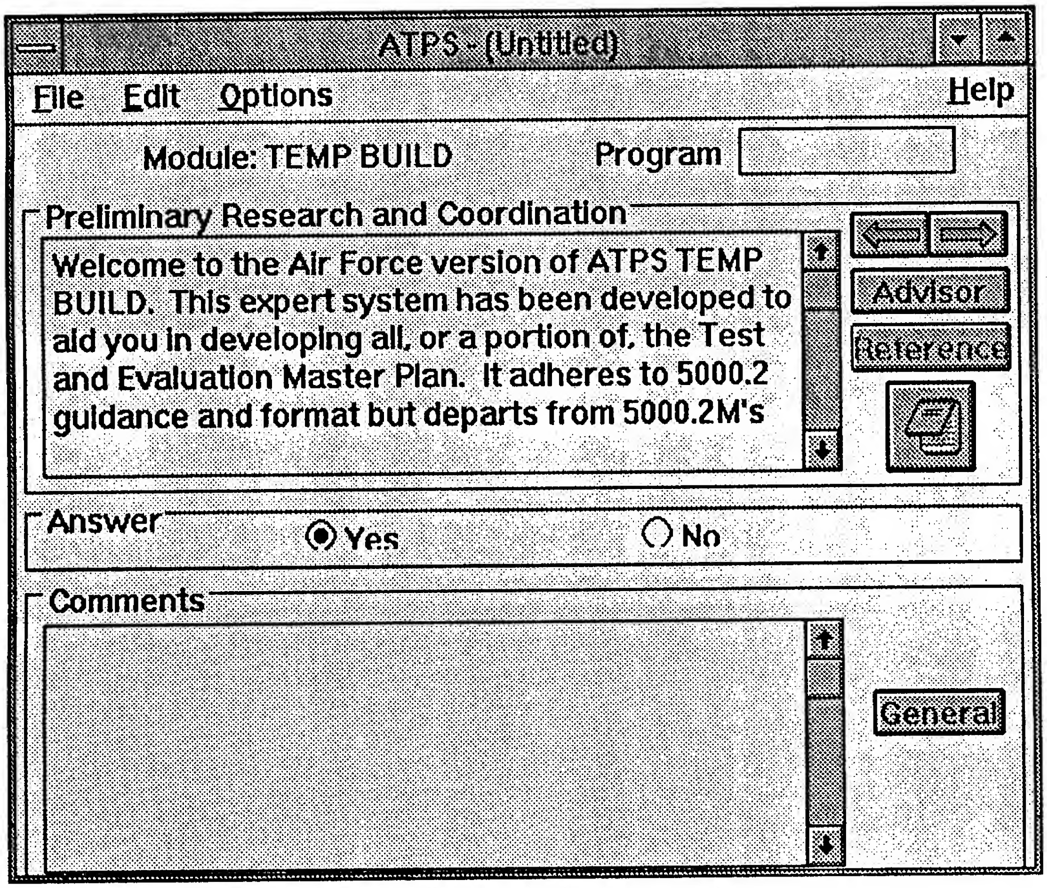


Figure 2

The buttons to the right of the Intelligent Checklist area allow the user to advance or back up through the checklist. The "Notepad" button (indicated by the small open notepad icon) allows the user to enter local guidance, tips, or notes that relate to the current checklist question. The purpose of the Notepad is to give individual sites limited ability to tailor the checklist to their needs. The Notepad file may be distributed to other ATPS users who will be able to view the notes. The "Reference" button is a cross-referencing feature that links the current checklist question to its reference in the DoD5000.2-M. If there isn't a reference available, as in the case of TEPRAM's harmonization checklists, the Reference button is low-lighted and not selectable. The "Advisor" button activates rules that find hints, tips or advice that expand upon the current checklist question. Figure 3 shows an example of the Advisor.

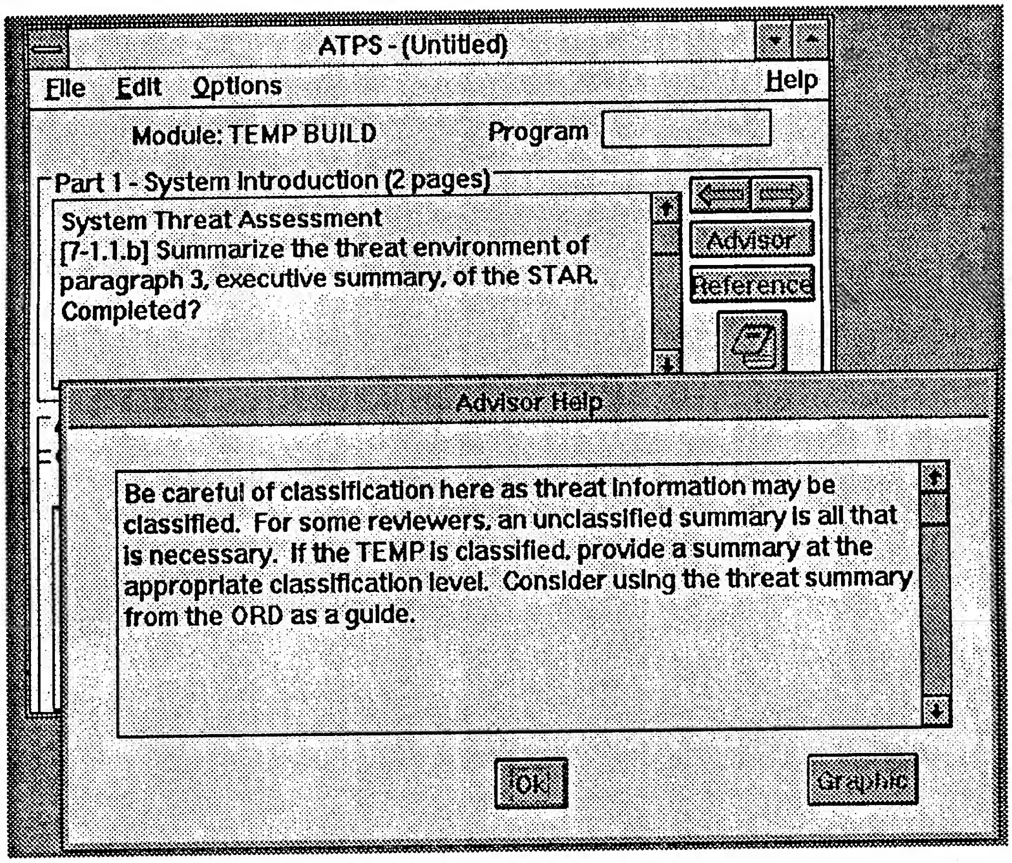


Figure 3

Referring back to the ATPS main screen, to the right of the Comments Editor, the "General" button activates an editor that allows the user to enter general comments regarding a session. These comments are not tied to any specific checklist question, rather, they are session dependent.

Handy editing features, such as cut, copy and paste, are available and active throughout ATPS. For example, portions of the hypertext help documents may be copied and pasted into the Comments Editor. The Advisor information may also be copied and pasted into the Comments Editor.

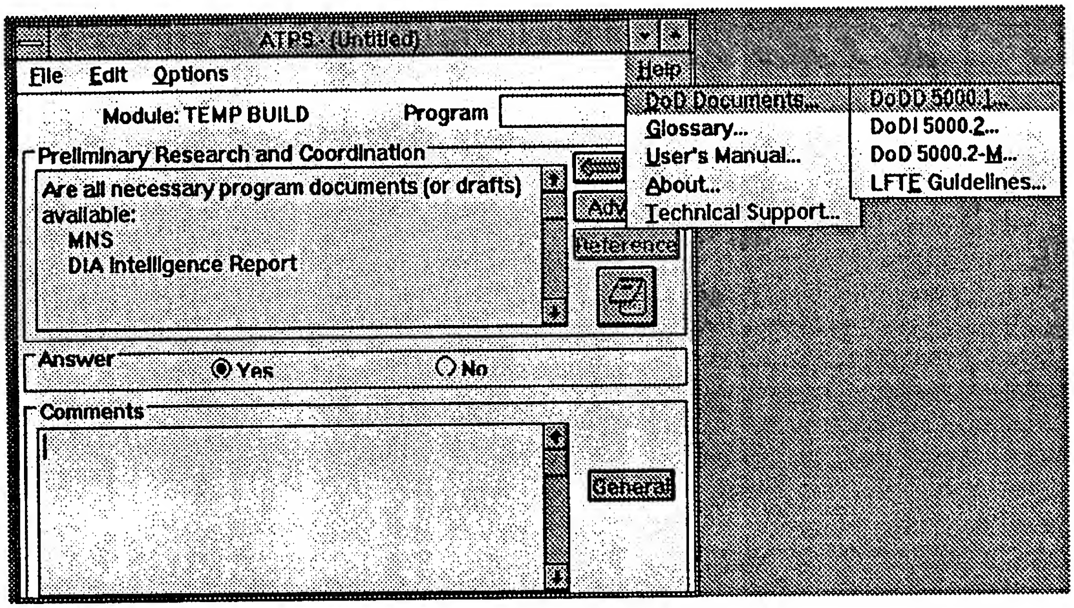


Figure 4

ATPS has an extensive on-line help facility. Figure 4 shows some of the help features available. The DoD 5000 series documents are available on line in hypertext format. Hypertext allows the user to search through a document in a non-linear format, much like reading a book from its index. A Glossary of terms is available containing several thousand entries. An on-line user's manual and phone numbers for technical support are also accessible.

Sessions may be saved and resumed at a later time. The ATPS software will resume a session where the user left off previously. Sessions may also be written out to an ASCII file which can be read by any standard word processor.

TEMP Build Module (TBM)

The TEMP Build Module's intent is to provide a consistent method to help the user build draft TEMPs in the DoD5000.2-M specified format. The TBM contains Service-specific guidance, as well as guidance from the DoD 5000 series and presents this information to the user through the use of expert-system generated checklists. Each checklist question is cross-referenced to the appropriate section of the DoD5000.2-M. The Comments Editor allows the user to enter in responses to the checklist questions.

Since each Service has their own "vocabulary", as well as differences in their TEMP building approach, the checklists were built to reflect these differences while still maintaining DoD 5000-level guidance. Figure 5 shows the session data that helps the system select the appropriate rulesets and rules to use in the TB session.

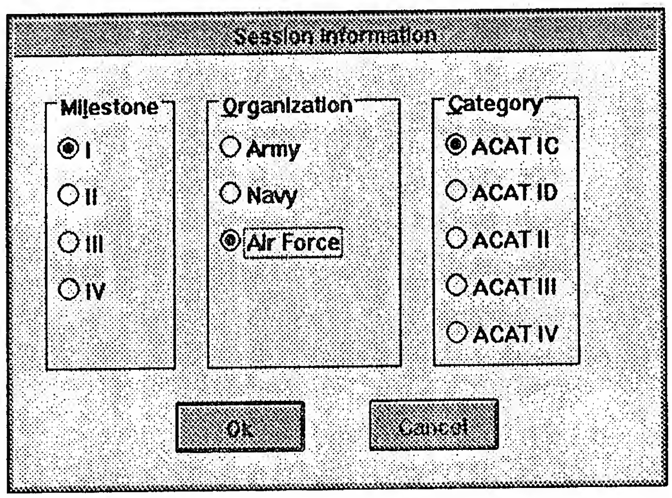


Figure 5

As the user steps through the checklist, his responses to the checklist questions become paragraphs of the TEMP. On completion of the TBM session, ATPS generates a draft TEMP, outputting the users comments in the format prescribed by the DoD5000.2-M.

T&E Program Risk Assessment Module (TEPRAM)

The TEPRAM goal is to provide the T&E community a way to achieve earlier involvement in the system acquisition process. This module allows the analyst to do two things: harmonize key acquisition documents (COEA, STAR, TEMP, ORD), and assess program risk from a T&E perspective. By ensuring that the key documents are harmonized, overall program risk is reduced

The assessment of program risk is an extremely challenging problem. Action officers currently assess risk in a variety of subjective manners. The common "stoplight" approach often does not provide the granularity required in assessing a realistic level of risk. TEPRAM leads the analyst through a series of questions, or an "interview", while maintaining some of the flavor of the familiar "stoplight" approach. At each question, the analyst chooses a risk indicator: red, yellow, green, or none (high, medium, low, or no risk)., and may also

enter explanatory comments. There are over 600 questions spread out among several sub topics to consider. At the end of the interview, TEPRAM writes out the analyst's session. The session is sorted "by risk," so that the analyst has an at-a-glance view of how many high, medium, and low risk issues he has identified. This approach provides the granularity previously missing from the traditional stoplight method. In addition, the TEPRAM approach provides a structure and a formal process for determining the root causes of risk, which have been missing from existing approaches.

TEMP Review Module (TRM)

This module's primary goal was to provide OSD with a tool to facilitate oversight of the TEMP review process. This module works to satisfy that goal by providing OSD and Service PO's and PM's with an expert system generated checklist that would serve as the standard to which TEMPs shall be reviewed. This helps to ensure more consistency in the review process, remove some of the "personalities" from the process and reduce training time for new TEMP reviewers.

The knowledge sources for TEMP Review were gleaned from extant paper checklists, experienced DT&E and DOT&E action officers, and guidance from the DoD 5000 series documents. The ATPS Working Group and Senior Advisor Group reviewed the knowledge going into the checklist.

A typical user has a hard-copy TEMP in hand as he goes through an ATPS TRM session. The TRM poses a question from the checklist and the user enters his response into the Comments Editor. When all of the questions generated by the expert system have been answered, the TRM allows the user to write his session to an ASCII file that may be read by any standard word processor.

Summary

ATPS is an analysis tool, designed to aid the human analyst, not to replace technical thought. It provides a standard baseline for TEMP development, risk assessment, and evaluation within a rich, easy-to-use, interactive environment. As DTSE&E moves forward, streamlining the acquisition process by implementing tools such as

ATPS will help to improve productivity and meet cost-cutting measures without sacrificing high-quality work.

Resources

Microsoft Windows, Windows SDK, Microsoft C 7.0, and the Microsoft Multimedia Viewer Publishing Toolkit are registered trademarks of the Microsoft Corporation.

RAL is a trademark of Production Systems Technologies, Inc, 5001 Baum Boulevard, Pittsburgh, PA 15213.

MPW is a trademark of Apple Computers, Inc. MPW C is available from APDA, PO Box 319, Buffalo, New York 14207-0319.

ATPS is government-owned software, and is available to government test agencies free of charge by contacting M. Scott Roth (703) 847-5595. The current PC release, ATPS 4.0 for Windows, contains TEMP Build, TEPRAM, and TEMP Review, the DoD 5000 series documents in hypertext form, a glossary of terms, and an on-line user's manual. The current Macintosh release, ATPS 3.0 for Macintosh, contains TEMP Review, and TEPRAM, the DoD 5000 series documents in hypertext form, and an on-line user's manual.

Acknowledgments

ATPS is supported and funded by OUSD(A&T)/DDTSE&E(A&SP), Richard Ledesma. The development of ATPS is and has always been a team effort. It could not exist without the support and guidance of the ATPS Working Group members and Senior Advisory Group. The authors would like to thank Dr. Adelia Ritchie, who conceived and initiated the ATPS effort and who implemented the unique team development approach; Dr. John Wiles who gave additional focus and direction to the product; and M. Scott Roth (PI), Kral Ferch, John Locke, and Dick Helmuth, all with the SAIC development team.

INSTRUMENTATION SUB-WORKING GROUP (ISWG)

Co-Chairs:
Teresa Telles and Mark Smedley

OFF THE SHELF TECHNOLOGY USED FOR SIMULATOR VERIFICATION AND ANALYSIS SYSTEM

Kevin J. Cassidy*
Julius C. Darden

Tektronix, Inc.*
Irvine, CA 92714

Westlake Systems Corporation
Westlake Village, CA 91360

Introduction

Verification of radar warning receivers (RWR) and radar jammers requires radar simulators that can create complex signals. When an RWR or jammer is being tested and does not respond correctly to the simulated signal, both the simulator and the EW system must be checked for correct operation. A third, independent, verification system is required. Due to the complexity of the radar simulations, an engineer or technician could spend days trying to measure and verify each threat manually. This paper describes a system based upon commercial off the shelf (COTS) hardware which can be used to record and verify radar simulation at a rapid rate. Radar signal data collection and analysis can be accomplished using high-speed digitizers and a powerful PC to quickly find system faults.

Independent verification of simulator signals is possible with a system that integrates COTS hardware and Microsoft Windows based software. Sampling of the signal pulses is accomplished with high speed digitizers. Digitized pulses are collected in a computer and displayed or saved directly to a hard disk. If the data is recorded, an entire simulation is captured in a "flat database" file on high capacity hard disks. After the data is collected, the system software allows the operator to efficiently sort through the data using time and frequency domain analysis tools resident in the computer. If additional analysis or archiving of the data is required, the system data can be exported to one of several standard formats. By taking advantage of COTS equipment's low cost and ready availability an effective radar verification system can be built quickly and economically.

RDR-4200 Hardware

The verification system described in this paper is the Westlake Systems RDR-4200 Impulse Recording System. The RDR-4200 was originally built to meet the requirements of a U.S. government customer who needed to record large numbers of radar pulses for analysis. The current system uses one of two Tektronix digitizers, the RTD-720A (see figure 1) or the TDS-684A (see figure 2). These digitizers have 3 dB analog bandwidths of 500 MHz and 1 GHz respectively. The RTD-720A samples data at up to two gigasamples/sec. The TDS-684A samples at up to five gigasamples/sec. With these wide bandwidths and high sample rates, downconverted, pulsed RF signals may be captured and recorded without aliasing. The RTD-

720A also has a very long record length of 4 megasamples. The 4 megasample record length is useful when a stream of pulses with complex modulations must be captured in a single record. Once an RF signal is digitized, its data is transferred to the RDR-4200 recorder. Data is transferred using a fast parallel bus on the RTD-720A or via the GPIB interface on the TDS-684A.

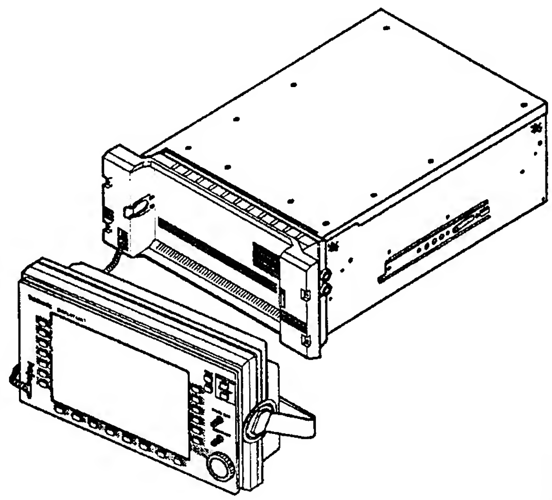


Figure 1. Tektronix RTD-720A Digitizer

The computer used in the RDR-4200 system is an IBM PC compatible. The computer's microprocessor is the latest high speed version of the Intel 80X86 family. Current systems ship with Pentium processors. The processor speed of the computer makes it possible to display digitized data on the system display in near real-time. In addition to a fast microprocessor, the computer utilizes the fastest IDE or SCSI hard disk currently available to store the digitized data. Another key component in the system's computer is an IRIG or GPS time code processor board. All data records captured with the RDR-4200 system have IRIG or GPS time stamps stored along with each data record. Finally because each customer's needs are unique, a custom designed RF deck is added to the system. This custom RF deck contains RF amplifiers, attenuators, AM detectors, and filters per the customer's requirements.

Data Recording

Recording radar pulses for long periods of time is the key feature of the RDR-4200. All the data from the RDR-4200's digitizer is transferred to the system computer's memory. After the desired number of records have been collected, the data is saved on the computer's hard disk. Data is saved in a "flat" file format that is unique to the RDR-4200 system (see figure 3), but can be easily converted to other formats with the current system software or other programs.

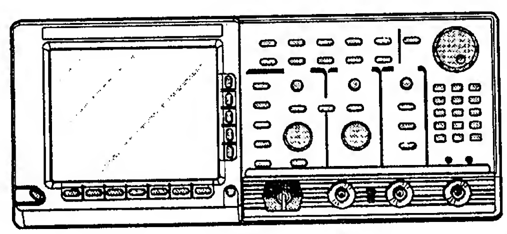


Figure 2. Tektronix TDS-684A Digitizer

The system's ability to digitize and record data (throughput) is typically fast enough to catch a live signal with the simulator running at full speed. There is no need to slow down or loop on a particular simulation. The throughput of the RDR-4200's recording system varies with the amount of data captured by the Tektronix digitizers. Short records such as 512 or 1024 point records can be transferred very quickly to the computer. The current system can continuously digitize and transfer 1024 point waveforms at 100 records/sec. When a longer length record (1-4 megasamples) is required because of signal modulations or long pulse streams, the ability of the

system to digitize, transfer data, and re-arm for the next acquisition decreases to about 10 records/sec.

One way to improve the overall system throughput and still capture data over long periods of time is to only digitize a signal when the signal is on. Tektronix digitizers have a mode of operation called "auto-advance." When the digitizer is running in auto-advance mode, it waits for a trigger. When a trigger occurs, the digitizer saves one record. Then the trigger is rearmed for another acquisition. The digitizer does not save another record until it receives another trigger. By only digitizing a set of pulses when the pulses are on, the memory of the digitizer can be used more efficiently. In addition, the system throughput can be improved by using short records (512-1024 samples), saving them in the digitizers memory until it fills, then transferring all the records to the computer. The software on the RDR-4200 computer handles multiple data records very easily and puts them all in a single database file. When the RTD-720A digitizer is run in auto-advance mode, the digitizer can sample a signal, put the signal in the digitizer's internal memory and rearm its trigger circuit in less than 10 μsec. Depending upon the PRIs and frame rates of the signals being output by the radar simulator, auto-advance can increase the RDR-4200's effective throughput by (1.) decreasing the time between successive data records and (2.) transferring data to the computer during simulation "dead times."

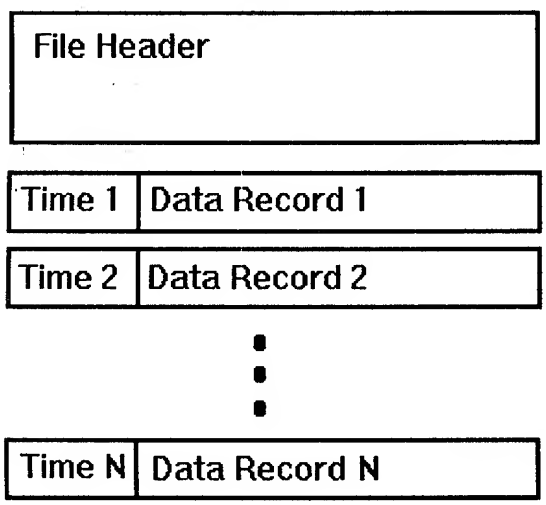


Figure 3 RDR-4200 Data File Format

The size of the RDR-4200 data files vary according to the number of records and the record length used in the Tektronix digitizers. However, with 1-2 GBytes of disk storage, the RDR-4200 can record hours of sampled data. Once the data is stored on the hard disk it can be analyzed by the RDR-4200's own data analysis software. In addition, many RDR-4200 users have successfully converted the RDR-4200 files to formats compatible with programs such as Mathcad and Matlab.

Each record in the RDR-4200 is time stamped with either an IRIG or GPS time stamp. The time stamp corresponds to the universal time value when the digitizer was triggered to capture the record. The RDR-4200 time code processor board with its own 10 MHz reference oscillator allows the system to resolve IRIG or GPS times down to 100 nsec. In addition the data within the digitizers is internally time stamped to a resolution of 500 psec. The RDR-4200 software includes the digitizer time stamp in its data files.

Recorded simulation data in a database can be recalled at a later time for analysis. This capability allows analysis by different analysts weeks, months, or even years later. The data could also be used to supplement a verification test's documentation. Files from the RDR-4200 data could be archived along with other test data. Lastly, a performance database of the simulator system could be built up over many years. Using archived data would allow the simulator maintenance organization to spot RF signal degradation over time. Using archived data, a system might be able to run a periodic self-test by comparing old database information to current samples.

RDR-4200 Software

The key to integrating the COTS hardware into an effective verification system is the RDR-4200 system software. All the programs in the RDR-4200 system are Microsoft Windows applications. The Windows' graphical user interface is well suited to analyzing the recorded radar data with many points of view. There are two primary modes to the RDR-4200 software, real-time and post recording analysis software. The real-time software includes the data recorder software and near real-time frequency and time domain displays. The post recording analysis software takes the recorded data as input and allows an operator to "visualize" the radar's output from many different views.

When the RDR-4200 is running in real-time mode, one of the ways that it can be used is in a data recorder mode. When used as a data recorder, the operator specifies the name of the data file, the number of data records to capture, and digitizer settings for the current test. When the operator initiates the data recorder software, the computer tells the digitizer to begin sampling the signal and transferring the requested data as it is acquired. Acquired data is combined with data from the time code processor and put in RDR-4200 data files. Multiple signal acquisitions can be put into a single data file or multiple files.

Another module in the RDR-4200 software includes a real-time frequency domain display. The RDR-4200 utilizes on board digital signal processing software to display an FFT of the captured data. The bandwidth of the FFT can be as wide as the 1 GHz analog bandwidth of the TDS-684A digitizer. Figure 4 shows an example of the frequency domain display of the RDR-4200. The bottom window shows a 4096-point FFT of the captured pulse data. The top window shows the frequency of the signal over time. The signal in this case was sweeping its frequency over several hundred megahertz. The frequency vs. time display can be used to verify frequency agile and linear FM simulations.

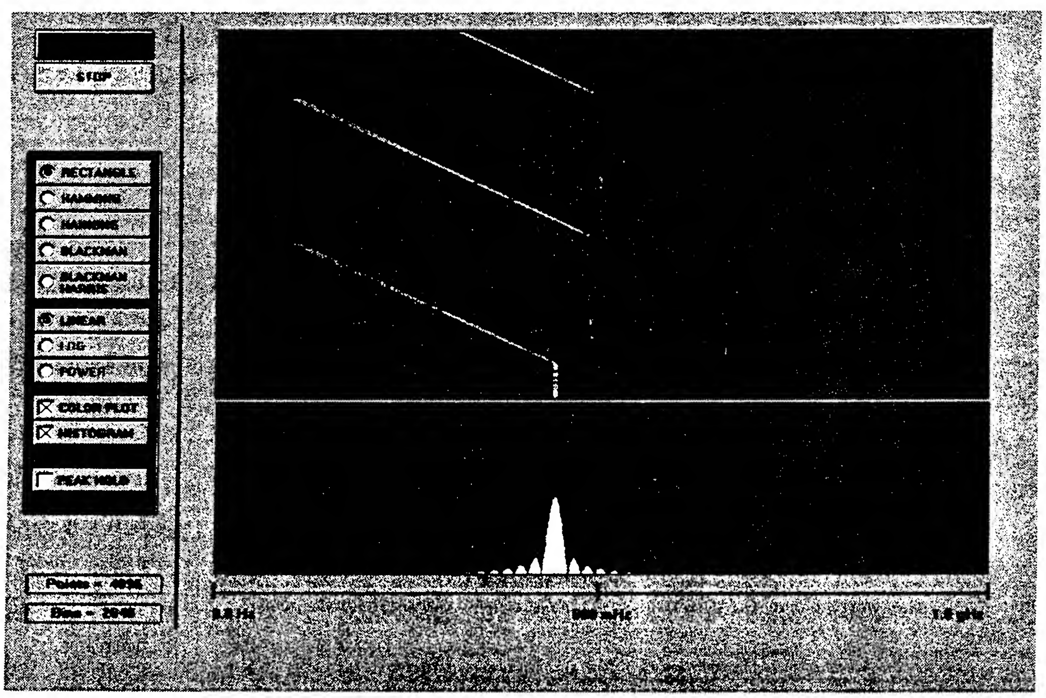


Figure 4. RDR-4200 Real-Time Frequency Domain Display

The RDR-4200 software also includes several time domain analysis tools. One tool is a real-time pulse viewer. This viewer is shown in figure 5. This viewer screen allows the operator to look at a signal with three different windows. One window is a standard single waveform view. Another is a time domain waterfall display. The third window is a color graded view of the amplitude of the captured pulses over time. With this pulse viewer it is easy to spot anomalies in a signal's pulse width or PRI by observing the signal over time.

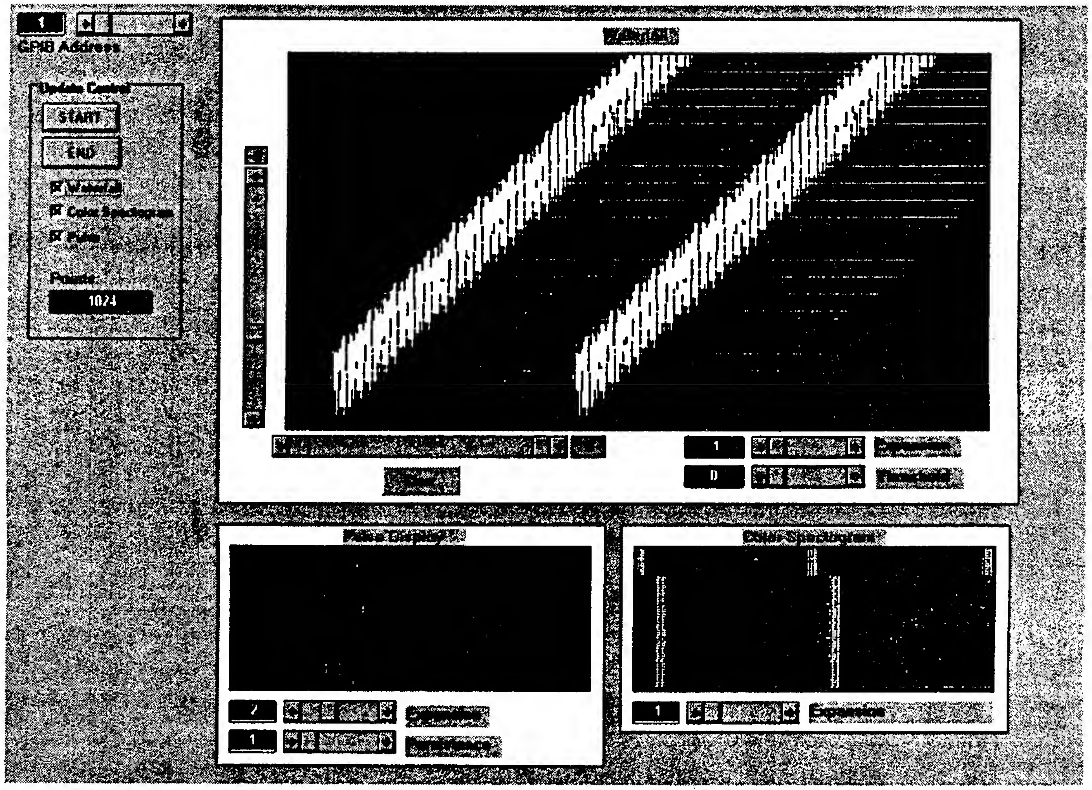


Figure 5. RDR-4200 Time Domain Display

Once data has been recorded, another set of graphical tools give the operator tremendous visualization power. For post recording analysis, the time and frequency domain tools described above are available as well as several others. One non-real-time function is called Smartvue. With Smartvue an operator can step through all the collected data records one by one. If a data record has an area that needs to be viewed in finer detail, the operator can highlight the area and zoom in on it (see figure 6). Another section of Smartvue is a pulse analyzer. The pulse analyzer is set up to measure long data records with multiple pulse widths and PRIs. The output from the pulse analyzer is a spreadsheet of all the pulse widths and PRIs found in the record. The spreadsheet data can be exported to a Microsoft Excel compatible file. Smartvue allows an operator to view and precisely measure each pulse width and PRI combination in a staggered PRI signal.

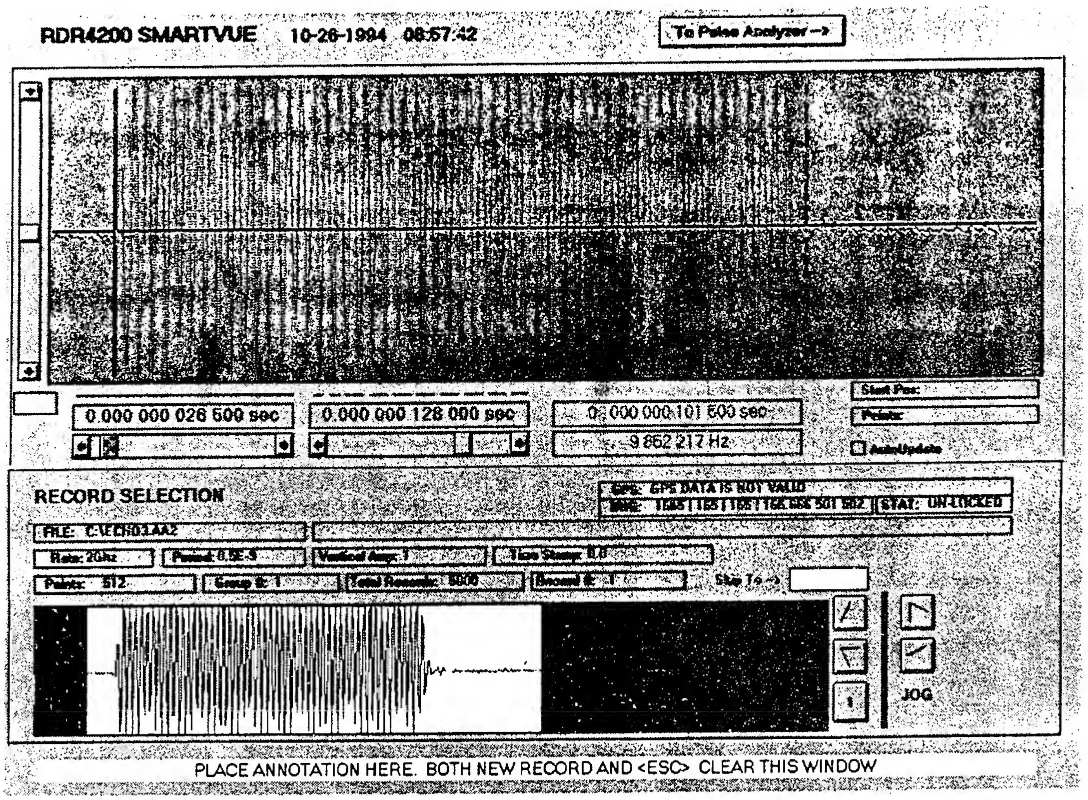


Figure 6. RDR-4200 Smartvue Screen

Conclusion

Today's radar simulators are capable of generating very complex signals. However, complexity can lead to errors. When an error occurs, a system for independent verification of the simulated signals is necessary. This paper has described a verification system built with commercial test equipment and computer hardware. The simulation data is sampled without aliasing while a simulation is running full speed.

The system described used COTS hardware and Microsoft Windows based software. Once the data is acquired, it is saved in a database file on the computer's hard disk or displayed to the operator in near real-time. Recorded data can be immediately analyzed or archived for later use as test documentation or system performance history. The Westlake Systems RDR-4200 takes advantage of commercial equipment's low cost and availability to create an very effective radar simulator verification system.

TEST SITE INTEGRATION

Gary A. Cooper and Gary P. Uhland
Aberdeen Test Center

Charles J. Rogers
SFA Inc.

Aberdeen Proving Ground, MD 21005
(410) 278-9468

ABSTRACT

Combat Systems Test Activity (CSTA) is one of the U. S. Army's major centers for development and technical testing of weapons systems. Over the past two decades CSTA has made a significant investment in the digitization of the ballistics and fire control ranges used in weapons system tests. The list of systems acquired in this digitization process includes: Weibel radars, Hadland cameras, video scoring of targets, systems to measure blast and chamber pressures, and flash x-ray imaging systems. This conversion to digital data acquisition systems has provided CSTA the capability to provide test data for review and analysis in near real-time.

CSTA has also made a significant investment in an activity wide local area network (LAN). This network has facilitated the collection and transmission of test data, both within the test ranges and throughout the activity. Test Site Integration (TSI) is the final building block in this digitization process. TSI is a generic system designed to integrate the digital test instrumentation and store test data at the various ballistics and fire control ranges located at CSTA. With this system, the test director and test sponsor have near real-time access to all of the digital data collected at a particular test range through a user friendly graphical user interface (GUI) running on a UNIX work station. The GUI allows the user to view, analyze and compare data from many rounds/runs, displaying text, images, plots, and digitized video.

TEST SITE INTEGRATION

Two major testing functions performed by CSTA are ballistic performance tests of direct fire munitions, and fire control systems tests of direct fire weapons systems. These tests are conducted at highly instrumented firing ranges, and generally involve the firing of a number of test rounds. The data generated from these tests are used by the test sponsors to characterize the performance of projectiles and propellants, and to assess the accuracy of the fire control systems of certain weapons.

A typical test range configuration consists of a firing barricade with a gun mount for stationary weapons, or a hardstand for mobile weapons systems. Flash x-ray systems are used to capture images of the round while it is still obscured by muzzle flash and smoke. Hadland digital cameras are positioned to capture images of the projectile just after muzzle exit. Weibel Doppler radars are used to determine the velocity of a test round, both at muzzle exit and during the flight

down range. The gun tube is instrumented to provide internal ballistic pressures, stress and strain measurements. Hadland digital cameras or digitized video images are used to determine the trajectory of a projectile during the flight down range. During fire control performance testing, digital instrumentation is configured to monitor the ballistic computer, gun tube pointing direction, and gunners aim point. These sources provide data for the evaluation of the fire control subsystems.

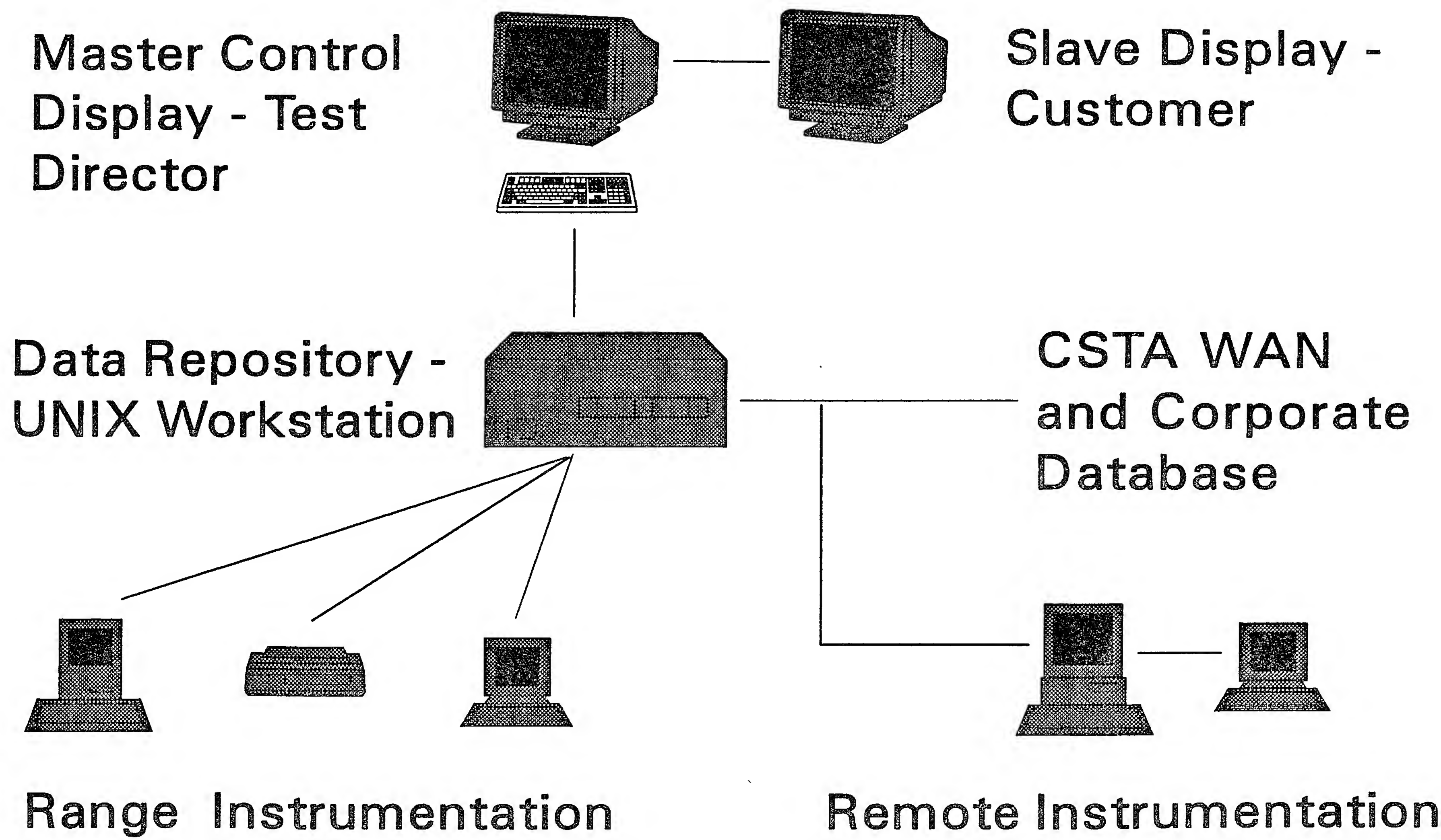
These test instruments are controlled by technicians from various groups within CSTA, and they are responsible for the data produced by that source. The test director oversees and coordinates with the data collection technicians and is responsible for the actual conduct of the test. A representative of the test customer is often present to ensure proper test conduct and review test data. As the speed and power of the various data collection systems has increased, it became apparent that a system was needed to integrate the data generated at the test site, and present this data to the tester and customer in an efficient and user friendly manner. TSI was designed to meet this need.

TSI consists of three major thrusts. The first thrust is the interconnection of digital instrumentation at the various ballistic and fire control test ranges using standard networking technologies. The second thrust involves using this network to collect and collate all of the digital test data acquired at a particular range. The third effort involves the development of a user friendly interface that will provide the test director and customer with information on the status of range instrumentation and provide access to test data in as close to real time as possible. One important goal was to design the TSI system to be as generic as possible. This will allow the system to be installed at a variety of test ranges and facilities with a minimum of modification. Figure 1 illustrates how TSI will be implemented at a typical test range.

CSTA has made great strides in implementing an activity wide Local Area Network (LAN). An extensive fiber-optic back-bone has been installed, providing fast, reliable data transmission capabilities between many areas of CSTA. As the facilities at the ranges are improved, ethernet sub-networks are being installed and connected to the fiber back-bone. As a result of this investment in network technology, test data can be transferred very efficiently, not only within the test site, but throughout the entire activity.

CSTA has also made a large investment in upgrading the data collection instrumentation that is used at the various test ranges. Most of these modern instruments are microprocessor controlled and provide data in a digital format. Test instruments that use film or video tape are being replaced by instruments such as the Hadland digital cameras or the digital flash x-ray system, which provide data in an electronic format. In order to minimize network connectivity, new test instrumentation was targeted to run on platforms that support the UNIX operating system. This was not possible in all cases, but the vast majority of data collection systems used at the activity are now UNIX or PC based.

One of the primary benefits of the network implementation and range instrumentation upgrades is the availability of test data for immediate review and post processing. Test directors



Master Control Display - Test Director
Slave Display - Customer
Data Repository - UNIX Workstation
CSTA WAN and Corporate Database
Range Instrumentation
Remote Instrumentation

Figure 1

can now have access to data in near real time. An organized system was needed to take advantage of the network, and ensure that this data was properly collected and stored.

TSI was designed to be a data repository for all the digital data collected at a test site. All of the digital instrumentation transfer test data to the TSI system running on the test directors work station. The data are then stored using a simple file and directory naming scheme. This scheme effectively archives the data so that it can be located and reviewed at a later time. This directory structure will also facilitate the loading of test data into the corporate database that CSTA will be implementing in the near future.

The basic TSI directory structure is a hierarchical structure with a directory corresponding to the test site at the top. Subdirectories corresponding to the particular tests conducted at the test site are below the test site directory. A subdirectory for each data source used for a particular test are below the test directory. The data files from these data sources are placed in their corresponding directories, named for the round/run that generated them.

In conjunction with receiving and storing test data, the TSI system notifies the users through a graphical interface that data for a particular round/run has been received from a particular test instrument. The user can choose to view this data at any time. Users can currently view numeric data, text, still images, and plots. Subsequent upgrades will include capabilities for viewing digitized full motion video. The following is a list of digital data sources an their statuses in regard to incorporation into TSI:

Data Source	Display Format(s)	Status
Ballistic Pressures	Text and Plots	Fully integrated.
Weibel Radar	Text and Plots	Fully integrated.
Hadland Digital Cameras	Images	Fully integrated.
Video Scoring	Text and Images	Fully integrated.
Digital Flash X-ray	Images	Awaiting delivery of system.
Meteorological Data	Text	Awaiting LAN connection.
Noise Monitoring System	Text	Awaiting LAN connection.
Digital High Speed Video	Images	Awaiting delivery of system.

The graphical interface also informs the user of the current status of range instrumentation. The test instruments send messages over the network, informing the TSI system of any changes in the status of the instrument. These messages are displayed on the graphical interface, eliminating the need for the test director to query his support personnel over the radio. Figure 2 shows a representation of the main TSI interface configured for a typical test.

The graphical interface also contains a Countdown Automated Procedure (CAP) that is used to initiate a system countdown, arm designated instruments at specified points in this countdown, and then fire a test round. The CAP displays a small countdown clock and an abort button on all of the UNIX systems used at a particular test range. Any of the test personnel that have access to a UNIX system on the range can use the abort button to stop the countdown. This

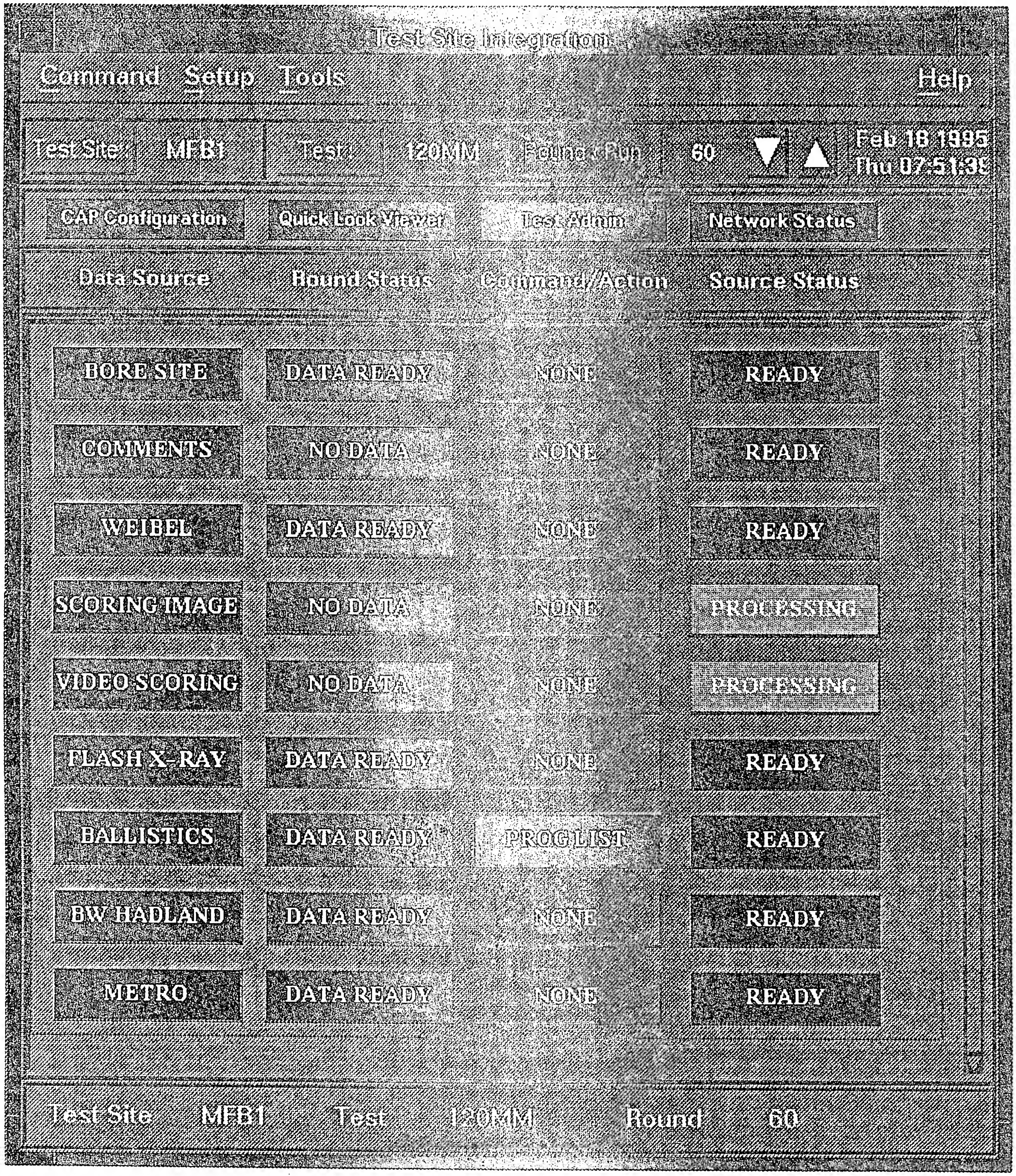


Figure 2 - TSI Main Interface

last second check increases range safety, and decreases the number of rounds fired with inactive or improperly configured instrumentation.

The standard communications protocols used for the TSI system are the Transmission Control Protocol and Internet Protocol (TCP/IP). TCP/IP are industry standard network communication protocols. TCP/IP is platform independent, allowing diverse instrumentation to communicate over the network. The Remote Procedure Call (RPC) protocols are used on top of TCP/IP and provide the actual vehicle for transferring messages and data within TSI. When using RPC, one computer system is configured as a server. This server is programmed to provide one or more services to remote clients. The scope of the services provided is only limited to the capabilities of the server computer. Other computers access this server as clients. The clients can request any of the services that the server is programmed to provide, transferring data and messages to and from the server if necessary.

The capabilities of RPC mesh quite well with the TSI system. The TSI work station is configured as the RPC server, and the test instrumentation are configured as RPC clients. The server is programmed to provide many services to the clients, one of which is the transfer of data files from the client to the server. The TSI system can also accept status messages from the instrumentation, and it can transmit current test configuration information back to the test instrumentation in order to keep such information up to date. As the TSI system expands, it is anticipated that the list of services provided by the TSI server will increase.

The Graphical User Interface (GUI) is an extremely important part of the TSI system. The GUI is what the user will use to interact with the range instrumentation and test data. The interface for TSI is a Motif based program running under X-Windows using the UNIX operating system. The user interface was developed using UIM/X, a Motif development environment available for most popular UNIX platforms.

When a test director starts the TSI program, he is presented with a window that has a toolbar and pulldown menus. By using the toolbar or menus, the user selects the test site being used for the current test. These test sites are presented in a popup selection box. After a test site is selected, the user is asked to select a test type from a popup list of test sites. After a particular test is selected, the program displays a row of buttons for each data source (test instrument) used for the test selected. The test director then selects a round or run from a list of previous round/runs executed for this particular test. If the test director desires, a new round/run name can also be entered.

The array of data source push buttons provides for most of the functionality of the TSI system. These buttons provide access to data from current or previous rounds/runs. The status message from all the data sources are displayed in these buttons. The post processing and analysis programs are also initiated in this section of the interface.

In addition to the data source specific buttons, the toolbar and menu system provide many useful features for the user. The test director also has total control of the actual test configuration. Individual test sites, tests, or data sources can be added to or deleted from a

configuration at any time. These utilities automatically configure the directory and file structure when new items are added.

Presently, TSI has the capability of displaying still images, text, numeric data and plots. Future upgrades will include video images. When it is possible, off-the-shelf software such as commercial image display programs and text editors are used for displaying test data. This maximizes the utility to the user and reduces the expense of developing similar programs in house.

CSTA has developed a Universal File Format (UFF) that is to be used for all data collected within the organization. The TSI system is designed to accept and read UFF data from the various data sources at the test ranges. All of the instrumentation software written for the initial implementation of TSI used this format.

UFF data files contain binary data. In order to display this data to the user, routines were written to extract the binary data and convert it to ASCII for display in the TSI interface. Any images collected at the test site are embedded in a UFF data file before being transferred to the TSI system. Routines were created to extract image data from UFF data files and display it.

Many data analysis routines are currently being rewritten or recompiled to execute on the TSI platforms. In situations where this is too expensive or time consuming, the remote shell feature of the UNIX/X-Windows operating system is used to run the analysis program on a different machine, displaying any output to the TSI terminal. By utilizing these features, the test director will have a large array of analysis tools available for his use.

CSTA plans to install TSI systems at all of the major ballistics and fire control ranges used by the activity. These systems will provide tester and customer with better access to both raw and processed test data. This increase in data efficiency will allow for timely changes in test scope and direction, providing both the tester and customer better products and services.

STREAMLINING TEST AND OPERATIONAL SYSTEMS

James LiVigni *
Timothy Conn
Fred Schreiber
Capt. Brian Fischer USAF
Capt. Richard Jernejcic USAF

EG&G Management Systems Inc. *
RATSCAT Operations
Holloman AFB, NM 88330

46th Test Group
Radar Target Scatter Division
Holloman AFB, NM 88330

Abstract

At any site or sites where multiple systems for test and data collection exist, an increase in efficiency, effectiveness and overall quality would be realized if the systems could be made similar from a planning and operational standpoint. If systems were made similar, operations, maintenance and training resources could be leveraged and more consistent quality assurance and configuration management practices put in place. Also if key elements of control, collection and processing could be standardized, each system could used to augment the planning and data processing for other systems. Finally, if each systems' specific control collection and processing functions could be encapsulated and made system independent, a method of test execution could be established so that tests could be planned and conducted identically from a functional standpoint regardless of the particular system implementation. This functional building block approach has been used successfully in Automated Test Equipment and while some testing and operations require significantly more user interaction these principles are also applicable.

By encapsulation of the system specific interfaces and controls using embedded processors with database driven command and control software, generic functions can be created. These functions provide a de-coupling of hardware dependencies from the overall test flow. In addition if the embedded subsystems can provide a standard output format and perform real time calibration, previously off-line process can be modified to perform real time diagnostics. The combination of these two features can significantly speed overall test time and allow for similar test and diagnostics methods on systems with dissimilar radar hardware.

I. INTRODUCTION

A. RATSCAT

The Radar Target Scatter (RATSCAT) Facility is the primary DOD facility for radar cross section (RCS). measurements. RATSCAT is comprised of two physically separate sites, Mainsite and RATSCAT Advanced Measurement System (RAMS) site. At RATSCAT, full-scale, flyable aircraft and missiles, operational vehicles, aerospace models, and miscellaneous targets can be accurately measured for radar target signature, both monostatic and bistatic, antenna gain and radiation patterns, and performance of active electronic systems. These measurements support weapons systems development, technology assessments, and related Department of Defense and U.S. Government sponsored efforts.

B. History

This paper is largely based on experiences from the Data Acquisition and Processing Improvement Systems (DAPS) project, an improvement and modernization program at the Radar Target Scatter Division (RATSCAT), Holloman Air Force Base, NM. The project was started in 1992 and initial operating capability (IOC) for the first system was April of 1995. The objective was to replace the data acquisition, control and processing subsystems from five separate Radar Measurement Systems, all having essentially the same mission, but each having very different hardware, software, and operational implementations. The replacement systems were to have common processors and peripherals, as well as a common hardware and software architecture. The goal of this effort was to leverage training and software improvements across all sites, streamline operations, simplify hardware and software maintenance and provide for future growth. The DAPS effort would in effect change the operations paradigm for each specific Radar Measurement System, and provide a site wide paradigm for control, collection, and processing of RCS data. The fundamental philosophy of the effort was "operations should dictate the hardware and software....not the hardware and software dictate operations".

C. Scope

This paper discusses the techniques used to encapsulate the hardware from each site and provide a standard method of test planning, test execution, data management and product creation. Also discussed are the utilization of common embedded processing components that can be configured to provide common data formats and pre-processing, regardless of the Radar system.

II. PROBLEM SYNOPSIS

At RATSCAT there are five major RADAR Measurement Systems. Each System was initially delivered to the Air Force based on specifications that dictated functional and performance requirements i.e. frequency, power, dynamic range, number of range gates, RCS product types etc. While the specifications did an admirable job defining what needed to be done, each specification let the vendors of each system figure out how to implement those specifications. .

While this separation of responsibilities is fine for a single system, the result when you have purchased five separate systems over a number of years is that the implementation varies significantly from system to system. As a result RATSCAT found itself with a number of systems utilizing HP1000s, VAX 11/780s, MicroVAXs, HP9000/800 and PCs for data acquisition and control. Each radar system's data acquisition and control subsystems utilized different user interfaces, media types, data formats, feedback, status mechanisms, etc.. In addition, each acquisition and control subsystem had different operational requirements based on the differences in the Radar systems themselves. These differences were exagerated further by the fact that not only was each radar measurement system different, but they each controlled different measurement ranges, with correspondingly unique field probing, calibration, target rotation, and target elevation systems.

The result of this diversity was that operations, training, and maintenance, along with any improvements in methods or techniques, could not be readily leveraged. Further complicating the problem was that a number of the existing processing systems were becoming obsolescent, and in need of replacement. For DAPS to be successful in providing a single set of hardware and software that would streamline operations this functional and temporal diversity of systems would have to be dealt with.

III. THE APPROACH

The approach taken to solve the problem was divided into three parts . Part one was to define a single set of physical things that needed to be done i.e. setting bit patterns, receiving bit patterns, merging bit patterns etc.. Part two was to define a method of describing operational functions and methods by which these functions could be executed independent of hardware implementation. Part 3 was to develop a system which could translate the functions into the physical entities that would implement them.

IV. CREATING A SINGLE SET OF INTERFACE REQUIREMENTS

The goal here was to find an all encompassing set of interface requirements that would span the existing interfaces. In addition, all the commands and related parameters used to control and communicate with each piece of radar and range equipment had to be collated. The interfaces were decomposed by type and protocol requirements, as well as by average and burst data rate requirements. Figure 1 gives and overall flow of the effort. The result of this effort was the creation of a single set of functional and performance interface requirements for the system. From these requirements, a single set of interface drivers was then created that was adequate for all sites.

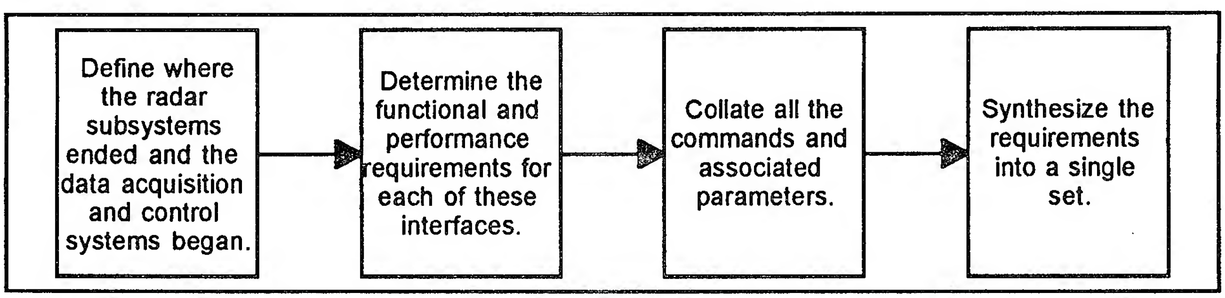


Figure 1 Interface Determination Flow Diagram

V. DETERMINING OPERATIONAL ELEMENTS

A. Finding A Common Thread

If a number of systems all have the same mission one can reasonably assume that at some level of abstraction they all do the same things. This was the initial premise of the DAPS analysis effort. It was quickly found that the one thing that all the sites had in common was the RATSCAT mission flow. This mission flow provided the top level operational sequence of events that were required, from the time a customer expressed interest in bringing a test program to RATSCAT until the final data product was delivered. This sequence of events was critically important to the effort; it had no coupling at all to a particular hardware implementation. It was this sequence of events that formed the basis for the effort to streamline the individual measurement systems was based.

B. Defining Common Procedural Elements of Test and Operations

By decomposing the mission flow requirements into lower and lower levels based on procedures that needed to be executed (regardless of the particular implementation) a hierarchy of procedural elements was developed. When put together, these elements allow the users and system support personnel to perform all the activities required to successfully construct and execute a test at any site, as well as generate all the required data products. Table 1 gives some examples and descriptions of these "atomic" operational elements. These elements have two distinctive features: 1) they are not hardware dependent; and 2) all elements of the mission flow (with respect to data acquisition and processing) can be made up of different combinations of these elements.

Table 1 Example Atomic Operational Elements

ATOMIC OPERATIONAL ELEMENTS	DESCRIPTION
Position Antenna	Position antenna in height and squint angle
Rotate Target	Rotate target in azimuth at specific speed over specific angle
Set Target Elevation	Set elevation of target on pylon
Set PRF	Set pulse repetition frequency
Set Range Gate	Set range gate in time and pulse width
Move File	Move date file from one media to another
Display Message	Display an alarm, warning or procedural message to operator
Set Frequency	Set frequency or start and stop frequencies
Start Collection	Start collection of data
Create Product	Create a display, plot or file
Archive Data	Store system parameters and data from test cut
Log Item	Log an item to a security or operations log file
Wait	Wait for completion of an event
Get Parameters	Query a device or devices for data

VI. TYING OPERATIONAL ELEMENTS TO PHYSICAL ACTIONS

A. Development of Test Planning and Execution Elements

Once data acquisition and processing function had been broken down into individual operational elements, two things needed be to accomplished. First, provide a method of organizing these procedural elements into different groupings to perform the larger operational procedures while maintaining the flexibility required to execute low level diagnostics. This method (known as test planning), once organized, provides a method of executing these elements on all the different systems at RATSCAT. This was labeled test execution.

To accomplish this, the team created of two levels of hierarchy, one procedural and the other physical, and then used an integral data base to tie them together. Table 2 gives brief descriptions of the procedural hierarchy developed. This organization follows the mission flow decomposition of the range users requirements into smaller and smaller executable elements. Table 3 outlines the physical hierarchy that ties the operational elements to physical devices.

Table 2 Test Execution Hierarchy

NAME	DESCRIPTION	EXAMPLES
Test Map	• Collection of scenarios that enables all tasks to be completed for a test program, test program phase or maintenance and diagnostics • Allows for scenarios to be added deleted and moved around scenarios can be copied from one test map to another or from a default set in the database	• Program 97-22 phase a test map, system response maintenance test map
Scenario	• Collection of procedures to perform a specific task • Procedures are executed sequentially • Automatic or manual sequencing	• Target a data collection, primary calibration HH pole, field probe
Procedure	• Collection of one or more function that do a specific physical thing • Function are executed sequentially • Automatic or manual sequencing	• 10 degree elevation, rotate 360 degrees, create suite of plots for target a, collect 370 degrees of data HV polarization
Function	• Atomic level of operation procedures • Provide lowest level of interaction for users under normal operation	• Rotate target, raise antenna 3, set radar polarization, plot RCS vs. azimuth, move data file

Table 3 Physical Activity Hierarchy

NAME	DESCRIPTION	EXAMPLES
Command	◦ Hardware specific series of actions that execute on a particular device ◦ Tied to site hardware/firmware of radar measurement system or to acquisition and control system for post processing activities ◦ Can Generate Actions	◦ Download Frequency table parameters, Set RF attenuation, Read Antenna Height
Action	◦ Lowest Level of RADAR and Range hardware control ◦ Actual Physical operation performed on Hardware ◦ Executed Immediately	◦ Set bit 3 address 2523, Read bit 7 at address 2012, Move 342 to address 1034

B. Test Planning Software Development Overview

In order to perform test planning software was developed to provide creation, modification and organization of test maps, scenarios and procedures. At RATSCAT, test engineers are principally involved in the creation of test maps for programs, while radar systems engineers are principally responsible for the creation of test maps for maintenance functions. By de-coupling the hardware from test planing, DAPS created an environment that allows the same sequence of events to be executed regardless of site. For example, field probing typically executes the following procedures at all sites; while the implementation may vary this from site to site, the sequence does not.

Set up Radar for Probe

Turn On Real Time Display

Start Collection

Raise Probe

Stop Collection

Lower Probe

The result of this organization of elements is a tremendous amount of flexibility to support the customer's needs, from R&D through production. Figure 2a shows the implementation of a test map which has scenarios composed of single procedures executing a single functions. While it may seem cumbersome at first, this one-to-one scenario to function mapping is quite useful to maintenance when exercising a specific portion of a radar measurement system. Figure 2b shows the structure of a test map that is geared toward production, where the test and data processing requirements are well defined, and the only thing that is going to change is the target configuration.

Test Map
(Low Level Implementation)

Set Channel 1
Set Channel 2
•
•
Set Channel 6
Set Plot Rectilinear
Set Axis Y
Set Scale Y
Set Axis X
Set Scale X
Set Title
RT Display
Set Target Azimuth
Set Target Elevation
Raise Antenna 1
Move Antenna 1 Azimuth
Move Antenna 1 Elevation
Raise Antenna 2
Move Antenna 2 Azimuth
Move Antenna 2 Elevation
•
•
Raise Antenna 6
Move Antenna 6 Azimuth
Move Antenna 6 Elevation
Set Target Azimuth
Set Target Elevation
•
•

2a

Test Map
(High Level Implementation)

Boresighting & Calibration
System Characterization
Target A Data Collection
Target B Data Collection
Data Package

2b

Figure 2 Realization of Test Matrix

C. *Creation of Functions*

The function is the lowest level of the operational system that the user sees. It provides the bridge between the low level operational elements and the site hardware. Functions provide that site independence of operations, that is one of the main goals of the DAPS effort. Creation of functions is accomplished by the development team working with each systems radar engineers to define what site specific commands and parameters are required to execute each operational element. The test execution software provides a list of utilities (called generic functions); all operational functions can be created from a list of thirty-three generic functions. Some functions pass data and commands to the embedded systems, while some are used for local data management and processing. Figure 3 gives some examples of this mapping of operational elements to physical actions.

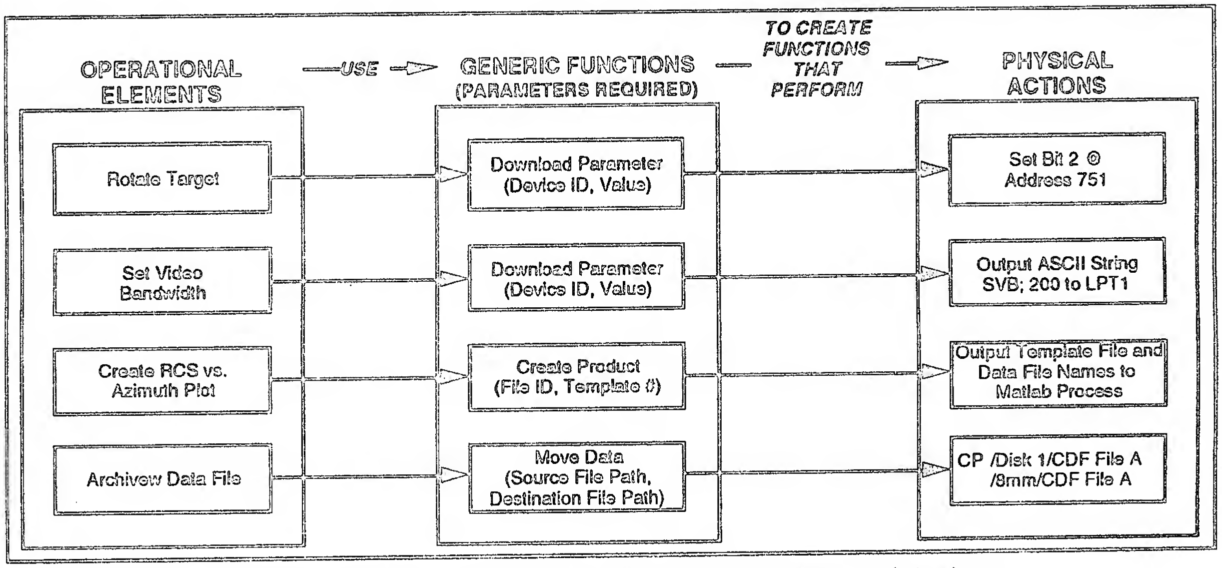


Figure 3 Operational Elements Translation To Physical Actions

VII. SYSTEM IMPLEMENTATION

A. *Overview*

The DAPS system is broken down into two subsystems. Test planning, test execution, data management, security management and product dynamics (changes to products, functions, database tables etc.) are part of the *Test Data System* (TDS). The TDS has all the system peripherals for plots logs, data products, etc. In addition, the TDS provides for storage of collected data. All the site specific hardware and firmware at each site interfaces to the *Embedded Resource System* (ERS). It is this embedded system that translates commands and status requests into physical actions and provides real time data for storage and display.

B. *Test Data System*

It is in the test data system that test maps, scenarios, procedures, and functions are created during test planning and then executed during test execution. Figure 4 gives an example of a test execution window . As each scenario is executed it appears in the test execution log portion of the window. This execution log can be annotated with comments and can be used as both a record of events as well as a method of obtaining metrics on system execution. Status on all pertinent system parameters can be accessed by pressing the appropriate button within test execution. These can than be iconified , moved, or closed as required. Security management, data management, and product dynamics windows and capabilities are not shown.

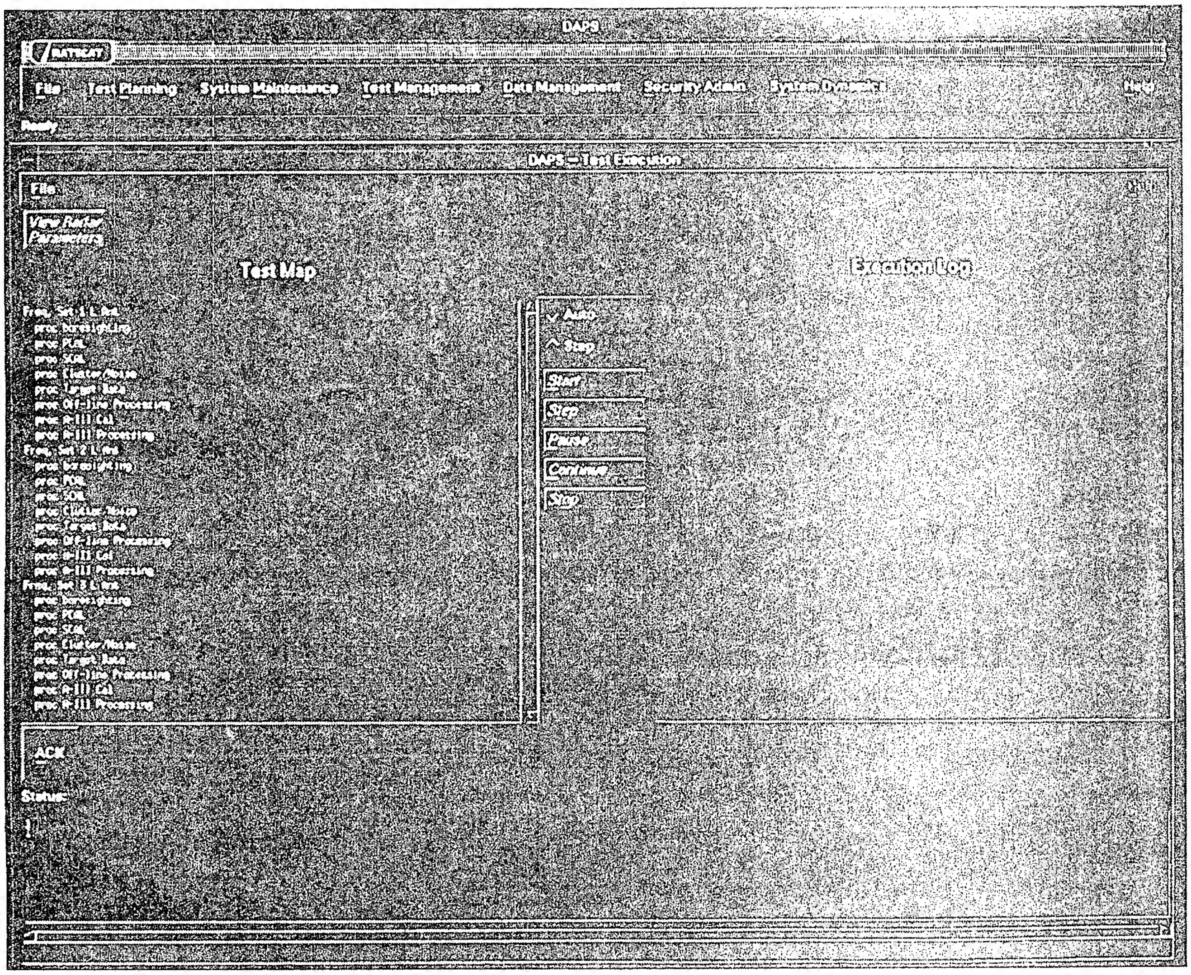


Figure 4 Main Window (Site Selected/Determined)

C. The Embedded Resource System

The mission of the embedded resource system is to translate commands from the TDS into site specific physical actions and to provide real time data processing for the data acquired. The system interprets commands and translates these commands into site specific physical actions. The ERS is made up of a core suite of control and communications software called the embedded resource manager (ERM), including device tables, interface drivers and data archive and display engines. Together these elements provide control and communications with all the site specific hardware and firmware as well as provide real time processing to the archive and display subsystems. Figure 5 is a simplified block diagram of the test execution portion of the DAPS system starting with the test map and ending with actions performed on physical devices.

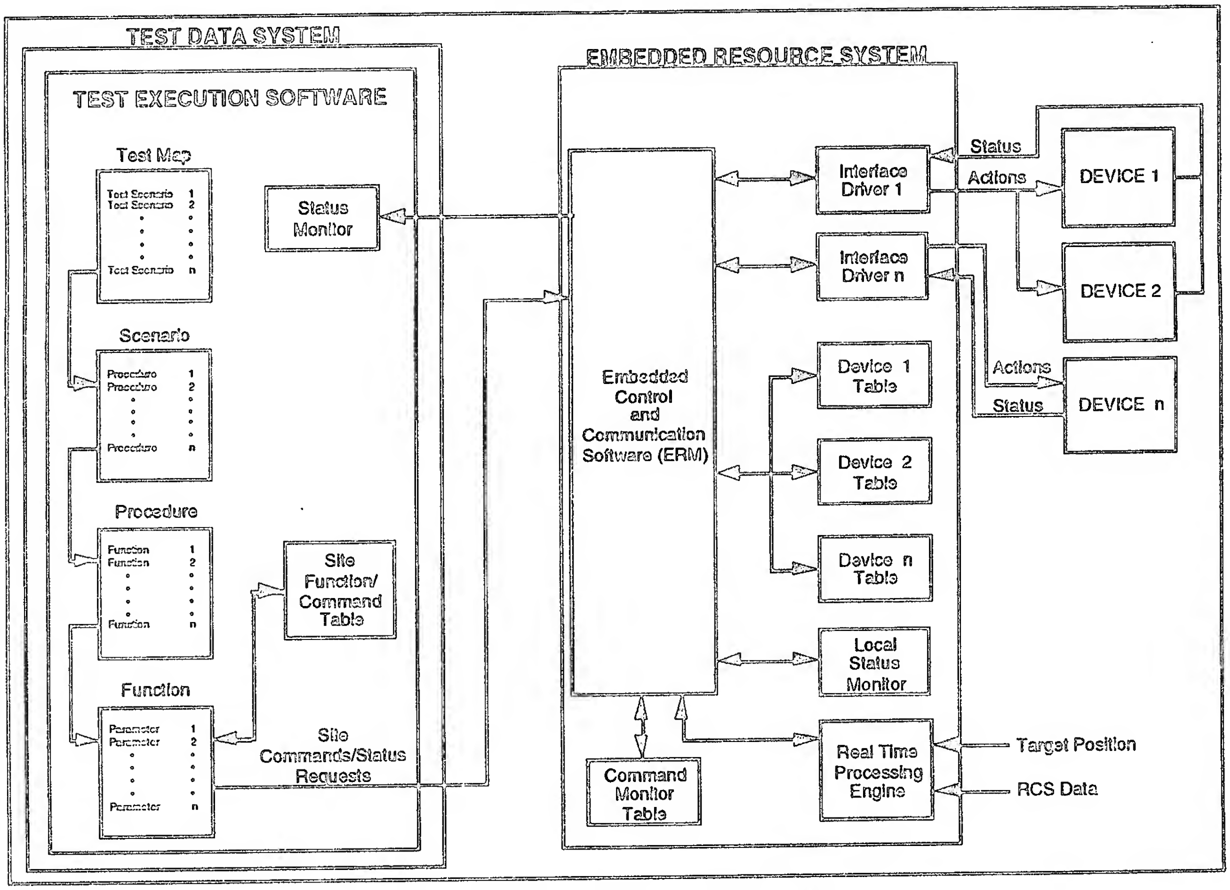


Figure 5 Test Execution Overview

D. *Command Monitor*

All commands and requests for status are routed by the ERM to the *command monitor* from either the TDS interface or a diagnostics interface. The command monitor utilizes command tables to check on the validity of input commands; once verified the embedded system will route the command to the proper device table. Typical control commands will have a unique message ID, command name, device ID, as well as the required parameters. The command monitor will determine if that device exists at that site and if the command has all the required information. If it does not it will return an unsolicited message to the TDS corresponding to the type of error encountered.

E. *Device Tables*

For each device that the ERS is required to communicate with and/or control there exists a *device table*. This table contains all the specific information required for control of that device. Once a command or request for status is decoded by the command monitor, the ERM will route the command to the appropriate device table. The device table is effectively a small copy of the overall system database. It checks if the value is within limits, and it will either utilize the command to vector to the appropriate action (bit pattern, ASCII string. signal, etc.) or it will calculate

the appropriate output. Once available, the device table will present the ERM both the response and the interface ID that the response need to be sent to.

F. Interface Drivers

Once a command or request for status has been processed through the device table, the ERM will route the action to the appropriate interface driver. For each system interface type there exists an *interface driver* that provides all of the timing and communications required for that particular interface. The performance and function of this driver was determined by the syntheses of all Radar system interfaces of that type into a single requirement that would meet all the different Radar systems requirements for that interface. In the case of status requests or unsolicited events (error messages, bus time-outs, warnings) the interface driver will present the status to the ERM.

G. Local Status Monitor

The *local status monitor* is effectively a table that converts low level status into appropriate ASCII messages. These ASCII messages are then routed by the ERM to the TDS. Based on the status message type, the message will appear in its own window (errors and warnings) or will appear in the status portion of test execution (normal systems modes and states). Otherwise it will be routed to one of the five parameter windows. The TDS is designed to send a request for status a periodic intervals as well as on request.

H. Real Time Processing Engine

One of the most important aspects of the DAPS system is its *real time processing engine*. This engine is made up of a quad digital signal processing board utilizing four TMS320C40s, and is referred to as the radar data pre processor (RDPP). The RDPP, along with custom "C40 based" Universal Logic Interface (ULI) Boards, essentially provide 8, 16, 32 and 64 bit bi-directional interface and processing of Radar Cross Sectional (RCS) data from the Radar measurement system. During data acquisition, combinations of input ULIs, along with the RDPP, convert the Radar's unique data format into the Common Data Format (CDF) that is the current standard in the RCS community. Once converted, the RDPP then provides output ULIs with both "raw" data for archival, as well as calibrated, motion compensated 1D and 2D processed data for real time display.

V. CONCLUSION

A. Program Synopsis

The DAPS effort has resulted in a robust architecture that is will meet the data acquisition control and processing needs at RATSCAT, both now and for the foreseeable future. An example of the systems adaptability from site to site was evidenced in the relatively small amount of software redesign between the first two system being deployed. The TDS software required only minor table changes and the ERS required a function change in the RDPP and a new

set of command, status and device tables. The remaining systems will require similar amounts of changes.

What did not change at all is test planning and execution. Both systems will conduct tests in a similar manner with all the hardware differences effectively hidden from test operations.

B. Recommendations

The DAPS effort required a significant amount of analysis and design – more than any single system upgrade would. If only one or two systems are to be made similar with no more new systems planned, this de-coupling effort has only marginal advantages. On the otherhand, there is a desire to provide standards of operations, data acquisition and control for a number of existing systems, this method of hardware encapsulation has numerous benefits in terms of leveraging manpower and providing consistent quality across the sites. This also allows for a system robust enough to meet future system needs with minimal changes

C. Advantages

Common procedures, nomenclature, training, and resources gives the test engineer a common pool of resources for test conduct. In addition, the ability to view the execution sequence via the execution log (Figure 4) provides metrics that allow creation of new operational elements to further streamline operations. For example, if an engineer finds that a number of procedures are always used in a certain way, they can be put into a single procedure. The same holds true for scenarios. As confidence in the system grows, more and more procedures with more and functions in them will be executed in automatic mode. All in all it is believed that this effort will provide RATSCAT with a state-of-the-art operational capability for the foreseeable future.

SANDIA NATIONAL LAB'S PRECISION LASER TRACKING SYSTEMS

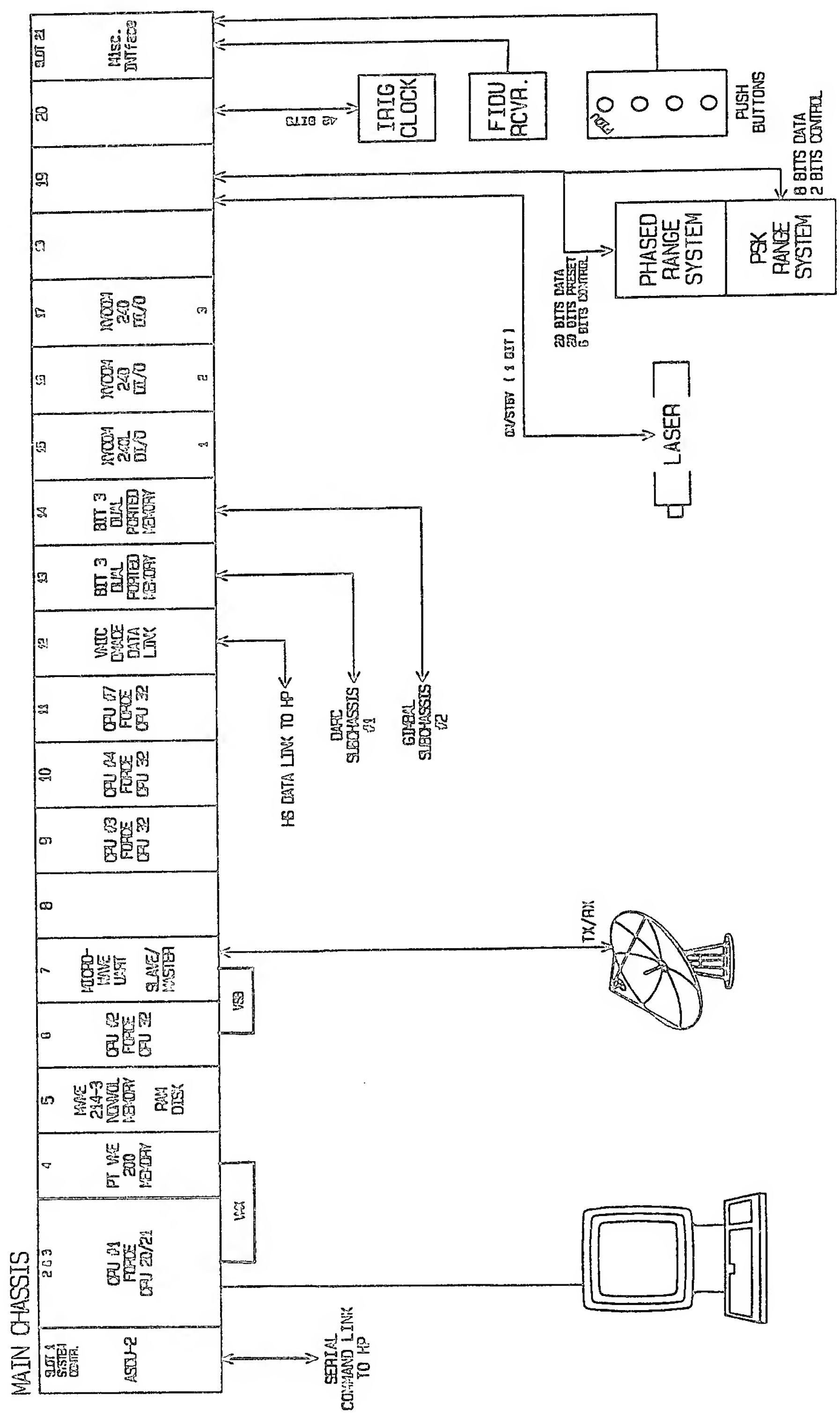
Duane Patrick

Sandia National Labs
Energetic and Environmental Test Department
Albuquerque, New Mexico 87185

Sandia Labs' mobile tracking systems have only one moving part. The double gimballed 18" diameter beryllium mirror is capable of constant tracking velocities up to 5 rads/sec in both axes, and accelerations to 150 rads/sec/sec in both axes. Orthogonality is < 10 microradians. The mirror directs the 488 and 514 nm wavelength CW laser beams to adhesive-backed reflective material applied to the test unit. The mirror catches the return beam and visual image, directing the visual image to three camera bays, and the return beam to an image dissector behind an 80" gathering telescope. The image dissector or image position sensor is a photomultiplier with amplifying drift tube and electron aperture and its associated electronics. During the test, the image dissector scan senses the change in position of the reflective material and produces signals to operate the azimuth and elevation torque motors in the gimbal assembly. With the help of 1 1/8" diameter azimuth and elevation galvonometer steering mirrors in the optical path, the laser beam is kept on the target at extremely high velocities. To maintain a constant return signal strength, the outgoing beam is run through a microprocessor controlled beam focusing telescope.

To produce real time three-dimensional position data, the tracker uses slant range and azimuth and elevation encoder readings. The ranging is accomplished by modulating the beams with electro-optical modulators, and doing a phase comparison between outgoing and reflected beams. The range information is accurate to within 6". The angular information coming off 18-bit binary electro-optical shaft encoders, is strobed into a minicomputer. along with an Irig time every millisecond. The encoder resolution is 24 microradians. The raw data is then desampled and reduced and used to produce trajectory data in a variety of media within an hour after the test. With these tracking systems, data, as well as closed loop servo control, pass through a real time control system. This microprocessor control system consists of seven single board computers residing on three inter-connected VME chassis, the main chassis and two subordinate subchassis. The main chassis (fig. 1) has five processors performing the following tasks:

1) Command interface with the operator, and data preprocessing and formatting.
2) Communications with source of slaving data and associated coordinate conversions.
3) Trajectory prediction calculations when tracking a target. When the target is lost the predicted path will be followed in hopes of reacquiring the target.
4) Control of the tracking electro-optics.
5) Control of the target ranging system.



MAIN CHASSIS
SLOT 1 SYSTEM CONTR.
ASCU-2
2 & 3
CPU 01 FORCE CPU 20/21
4
PT VME 200 MEMORY
5
VME 214-3 NONVOL MEMORY
RAM DISK
6
CPU 02 FORCE CPU 32
7
MICRO-WAVE USART
SLAVE/MASTER
8
9
CPU 03 FORCE CPU 32
10
CPU 04 FORCE CPU 32
11
CPU 07 FORCE CPU 32
12
VMIC CMADE DATA LINK
13
BIT 3 DUAL PORTED MEMORY
14
BIT 3 DUAL PORTED MEMORY
15
XYCOM 240L DI/O
1
16
XYCOM 240 DI/O
2
17
XYCOM 240 DI/O
3
18
19
20
SLOT 21
Misc. INTface
SERIAL COMMAND LINK TO HP
TX/RX
HS DATA LINK TO HP
DARC SUBCHASSIS #1
GIMBAL SUBCHASSIS #2
OV/STBY (1 BIT)
LASER
20 BITS DATA
20 BITS PRESET
6 BITS CONTROL
PHASED RANGE SYSTEM
PSK RANGE SYSTEM
8 BITS DATA
2 BITS CONTROL
48 BITS
IRIG CLOCK
FIDU RCVR.
PUSH BUTTONS

Figure 1

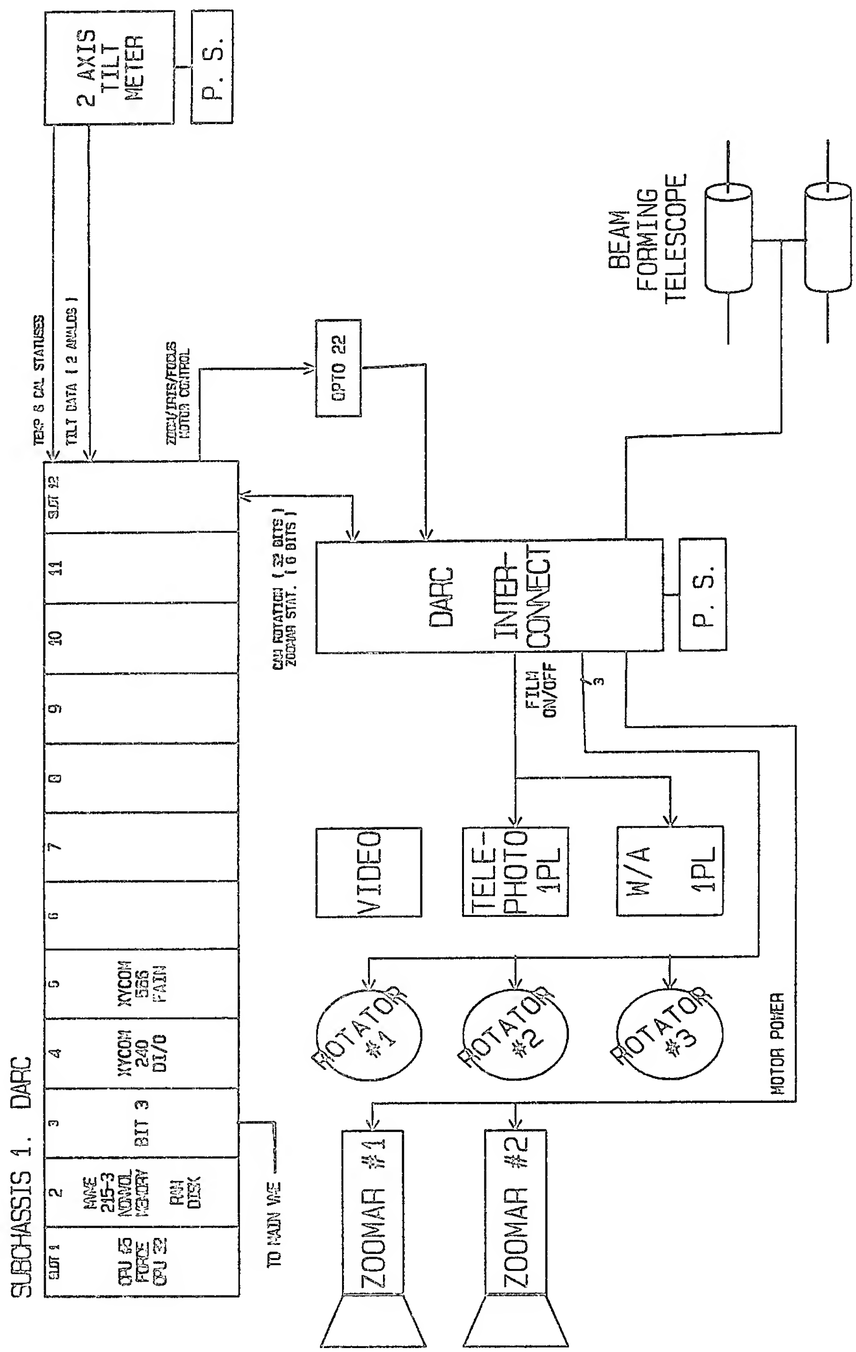
Subchassis #1 (fig. 2) controls all camera, lense, rotator, and laser beam expander functions. Tracking from a gimballed mirror results in image rotation. The camera stages are counter-rotated to maintain a level field of view. Photometric coverage includes shuttered video with displayed timing to the millisecond, as well as high-speed 16mm film with numeric timing, for more detailed analysis later. The laser beam expanders can converge or diverge the two beams to keep a constant return signal strength from the target. The green beam is used for coarse ranging and tracking. The blue beam is used for intermediate and fine ranging.

Subchassis #2 (fig. 3) performs gimbal control and tracking functions. Gimbal movement can originate from joystick or test definition commands, or from tracker error signals generated by the tracking image dissector. In either case, the position error is determined and the gimbal is driven to null that error. Gimbal response and serrvo compensation are both determined digitally.

The laser tracking systems have the ability to acquire targets by initial lock-on, transfer, acquisition on the fly, or joystick to auto track. Transfer tests require the system to lock on to a surrogate target transferring to the real target when it obscures the surrogate. This allows high - speed acquisition of targets with no scanning by the image dissector. Acquisition on the fly enables the system to lock on to a target passing through the diverged laser beam pattern. Joystick to automatic laser tracking is useful in manually following slower targets until laser lock is achieved.

In the last twenty-five years at Sandia Labs' Coyote Canyon Test Complex's 10,000-ft. sled track and 5,000-ft. aerial cable test facilities, Sandia's laser trackers have supported hundreds of test projects. Ranges to 25,000-ft. and velocities to 6,500 ft/sec. have been supported by a single tracker. Some of the test programs supported were WAM, SADARM, SFW, DAMOCLES, DIRCM, MAWS, and ATIRCM.

For tracking rocket sleds, aircraft and helicopters, missiles, and submunitions, the introduction of Sandia's laser trackers as a single station solution for Time Space Position Information (TSPI), has cut costs and data reduction time significantly as well as improving data quality. The laser trackers have proven to be reliable, state of the art tracking systems capable of supporting a wide range of test requirements.



SUBCHASSIS 1. DARC
SLOT 1
2
3
4
5
6
7
8
9
10
11
SLOT 12
BIT 3
XYCOM 240 DI/O
TO MAIN VME
TEMP & CAL STATUSES
TILT DATA (2 ANALOG)
ZOOM/IRIS/FOCUS MOTOR CONTROL
2 AXIS TILT METER
P. S.
OPTO 22
BEAM FORMING TELESCOPE
CAM ROTATION (32 BITS)
ZOOMAR STAT. (16 BITS)
DARC INTER-CONNECT
P. S.
FILM ON/OFF
3
VIDEO
TELE-PHOTO 1PL
W/A 1PL
ROTATOR #1
ROTATOR #2
ROTATOR #3
MOTOR POWER
ZOOMAR #1
ZOOMAR #2

Figure 2

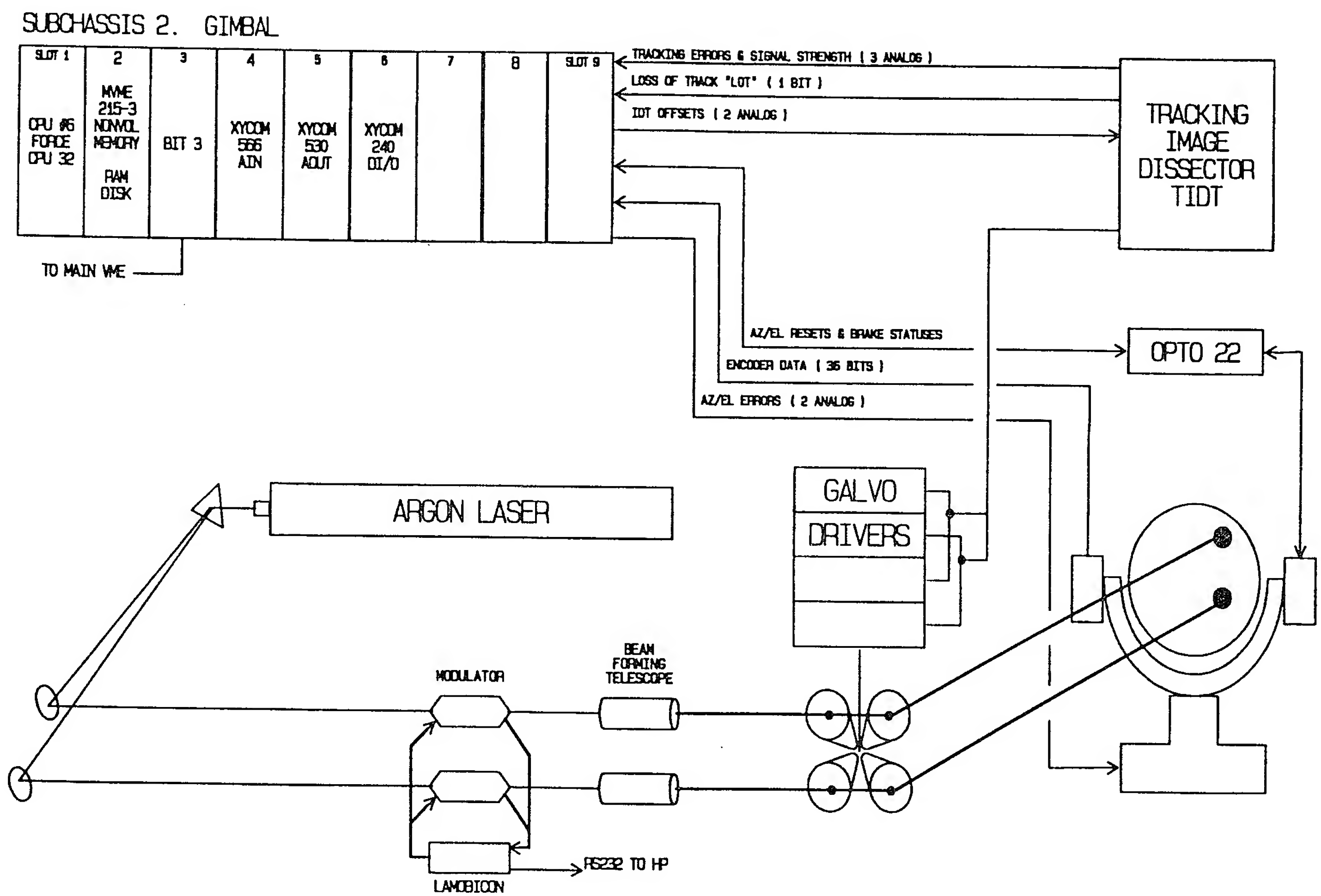

SUBCHASSIS 2. GIMBAL
SLOT 1
CPU #6 FORCE CPU 32
2
MVME 215-3 NONVOL MEMORY
RAM DISK
3
BIT 3
4
XYCOM 566 AIN
5
XYCOM 530 AOUT
6
XYCOM 240 DI/O
7
8
SLOT 9
TO MAIN VME
TRACKING ERRORS & SIGNAL STRENGTH (3 ANALOG)
LOSS OF TRACK "LOT" (1 BIT)
IDT OFFSETS (2 ANALOG)
TRACKING IMAGE DISSECTOR TIDT
AZ/EL RESETS & BRAKE STATUSES
ENCODER DATA (36 BITS)
AZ/EL ERRORS (2 ANALOG)
OPTO 22
ARGON LASER
GALVO DRIVERS
MODULATOR
BEAM FORMING TELESCOPE
LAMOBICON
RS232 TO HP

Figure 3

DATA ANALYSIS SUB-WORKING GROUP (DASWG)

Co-Chairs:
Kathy Render and Al Johnston

Application of the Avionics Data Visualization Integration System Environment to Different Test Domains

David Calloway
David.L.Calloway@cpmx.saic.com

SAIC
429 S. Tyndall Parkway, Ste H
Panama City, FL 32404

Abstract

This paper describes various applications of the Avionics Data Visualization Integration System Environment (ADVISE), an on-going development effort sponsored by WL/AAAF-2 under the Embedded Computer Systems Readiness (ECSR) contract. The emphasis in this paper is to show how the different tools in the ADVISE system can benefit several different types of test organizations. An overview of the ADVISE hardware and software components and a description of the key ADVISE technology areas are followed by specific examples of applying the ADVISE system in various test contexts.

1. Introduction

The ADVISE system advances the state of the art in avionics data analysis and scientific visualization through several key technology areas:

- Multi-Media Displays
- A Data Analysis Expert System
- Data Sonification
- Storage Management
- Hypertext User Manuals

In order to support and demonstrate each of these key technology areas, the ADVISE system integrates several leading edge hardware and software components into a cohesive environment for advanced avionics data analysis and scientific visualization. The ADVISE hardware platform is a high performance Digital Alpha workstation in a rack-mount, transportable enclosure. The software tools include:

- Oracle database and the ADVISE-enhanced EVADE system
- Visual Numerics PV-Wave data analysis software
- Archived Data Reduction System (ADRS)
- Internet-compatible hypertext browser
- NASA CLIPS expert system shell

After presenting a description of the key technology areas and the supporting hardware and software components that comprise the ADVISE system, this paper discusses how the resulting system can be applied in different test environments, including:

- Avionics software support facilities,
- Developmental Test and Evaluation (DT&E) facilities,
- Operational Test and Evaluation (OT&E) facilities,
- Radar test facilities, and
- Air-to-air and air-to-ground missile test facilities.

The facility descriptions are intentionally generic, but may be easily customized to fit specific facility requirements.

2. Key ADVISE Technology Areas

The ADVISE system advances data analysis and visualization technologies in several directions. These advances include multi-media displays, a data analysis expert system, data sonification, storage management, and hypertext user manuals. Each of these technology advances are described in the following paragraphs.

2.1.1 Multi-Media Displays

ADVISE multi-media displays consist of digitized video sequences that are played back in concert with other signal analysis displays. The ADVISE system includes a video digitizer to input and store any number of video sequences. The user synchronizes video displays with signal displays by identifying one or more key frames in the video and assigning a time code to the selected frame(s). The ADVISE system can then interpolate and extrapolate time codes for all other frames in the video sequence. After establishing frame times, the test engineer can select any video sequence and any number of traditional data displays (e.g. strip charts, 3-D animations, etc.) and the ADVISE system will play synchronized displays of video with the traditional analysis displays. The analyst can control the playback using digital analogs of standard Video Cassette Recorder (VCR) buttons, including Play, Pause, Stop, Fast Forward, and Rewind. These buttons control the sequencing and playback of the video stream, and the corresponding analysis displays are updated at the corresponding rates. The user can also identify a specific data variable/time and ADVISE will display (or play back) the corresponding video frame(s).

2.1.2 Data Analysis Expert System

The ADVISE data analysis expert system performs two primary functions. First, it analyzes avionics data to identify key events in the data. Second, the expert system knows the relationship between data sources and analysis/visualization techniques and can use this knowledge to recommend appropriate analysis and visualization techniques to the user based on the signals this user is currently loading into the system.

Key events can consist of problems or anomalies detected in an avionics data stream (e.g. lost track), unusual events that may be of interest to an analyst (e.g. weapons launch), and specific time "hacks". The ADVISE expert system detects these events, logs them in the ADVISE database, and notifies the user of any events contained in a signal stream being analyzed. The

logic for detecting anomalous and unusual events is based on the Multi-Source Interface Controller (MUSIC) system, a sister project to ADVISE that is also funded by WL/AAAF-2.

The ADVISE data analysis expert system is also familiar with the inter-relationships between various data elements in an avionics data stream and can recommend data analysis functions and data visualizations appropriate for these signals. The ADVISE expert system also recommends data sources to access based on up to three other signals recently accessed by an analyst.

2.1.3 Data Sonification

Sonification is the conversion of a generic signal into sound. A visual display can be enhanced by using the analyst's sense of hearing to relate additional information. For example, unusual events may be indicated by sounding a particular tone. This process can be extended further by mapping a continuously changing signal value into a corresponding sequence of sounds. A good example of this technique is the missile "tone" that is generated by the IR seeker in an AIM-9 air-to-air missile. This tone, fed back to the earphones of the pilot, provides instant feedback on the strength of the signal being generated by the seeker. This in turn gives the pilot the information he needs to know whether or not he is tracking a target with sufficient strength to warrant a missile launch. Using sound to relate this information prevents the pilot from having to take his eyes off the out-of-the-cockpit environment to read a dial, thus minimizing distractions at a critical point in a mission. ADVISE employs a similar approach to enable software support engineers to assimilate more information about a system being analyzed without being overly distracted with having to synchronize multiple visual displays. This is primarily an adjunct to visual displays, rather than a replacement for them, and can be used to add a significant measure of understandability to many scientific visualizations.

2.1.4 Storage Management

The ADVISE database stores the names and locations (tape ID and record indices) of all data files collected during a test. This information can be reported to the user to enable him to locate data from a particular test, and may also be queried by the data reduction tool to automatically load the appropriate tape if it is currently available in the ADVISE tape jukebox.

The ADVISE system also stores key signal information in an on-line database. It is not reasonable to store *all* signal data in a comprehensive database (there is simply too much data for this approach to work). However, a small subset of all collected data often holds the most important information content. This is the information ADVISE attempts to capture and store in the database. As mentioned in the expert system section, ADVISE contains an intelligent agent to identify and extract key events in an avionics data stream. These events are then recorded in the ADVISE database. Meanwhile, conventional magnetic disk and high capacity tape jukeboxes provide cost effective storage of the large volumes of complete signal data collected in a typical avionics test. The event indices that are stored in the ADVISE database provide the key information needed to locate and extract signals of interest.

Finally, the ADVISE database holds environmental data that describes where, when, and what equipment/personnel participated in a particular test. In fact, the ADVISE database is an outgrowth of the ECM Vulnerability Assessment Data Encyclopedia (EVADE) which was produced by Georgia Tech for WL/AARM. The EVADE database cataloged this environmental

data, but did not include the enhancements added by the ADVISE system to store key data extracts and the names and locations of all data files associated with a test.

2.1.5 Hypertext User Manuals

Most avionics software support facilities rely on relatively voluminous sets of documentation for a system under test. This documentation is often difficult to use because there is so much of it, and because the cross indices relating various parts of the documentation are incomplete or non-existent. In addition, a transportable data analysis environment is often hampered by a lack of sufficient documentation.

The ADVISE system remedies this situation by supporting on-line hypertext documents. After information from a conventional document is scanned, optical character recognition (OCR) is peformed on the scanned image to recapture the original ASCII characters, and the text is then integrated with scanned figures and tables to produce an equivalent on-line version of the manual. ADVISE hypertext documents are coded in the Hyper-Text Markup Language (HTML) that has become the standard for publishing information on the internet's world-wide web. This standard supports all the features required to create an on-line document, and has the added feature of compatibility with web servers that would enable any number of engineers in a facility (or anywhere in the world for that matter) to access a single, configuration-controlled version of a document or user manual.

On-line user manuals supported by the ADVISE system improve analyst efficiency by making the information instantly accessible at the engineer's terminal. The engineer can also easily create on-line book marks that he can return to with a simple mouse click. In addition, many web browsers permit individual users to add annotations to any on-line document. These annotations are then displayed whenever the engineer returns to the particular page in the manual. This is analogous to the user writing notes in the margin of a paper document.

On-line documentation is especially beneficial to novice users. The hypertext links enable these new users to immediately jump to the definition of an unfamiliar acronym or term, for example, and then easily return to the original context. Other links can instantly transport the user to a reference providing additional information on a topic. Although hypertext technology is now commonly available (as witnessed by the explosive growth of the world-wide web), the ADVISE system represents one of the first applications of this technology to an avionics analysis and visualization environment.

3. ADVISE Tools

The ADVISE system consists of several different hardware and software tools integrated into a single, cohesive environment. The hardware tools provide a state-of-the-art, high-performance platform on which to execute the software that gives ADVISE its fully integrated, user-friendly capabilities.

3.1 ADVISE Hardware

The ADVISE hardware includes a high performance, state-of-the art engineering workstation, a 3-dimensional graphics display subsystem, 4 GB of on-line disk and over 40 GB of near-line (jukebox) digital tape in a hierarchical storage management system, a MIDI audio system to support the sonification of signal data, and an ethernet Local Area Network (LAN).

3.1.1 Engineering Workstation

The ADVISE workstation is a DEC Alpha 3000 model 900, currently the fastest deskside workstation in the industry. This workstation is powered by a 64-bit, 275 MHz Alpha AXP RISC processor and performs 189 SPECint92 and 264 SPECfp92. It is based on a TurboChannel system bus operating at 100 MB/sec, and is mounted in a rack-mount enclosure for transportability in WL/AAAF-2's Mobile ECS Readiness Research Facility (MERRF) vans.

3.1.2 Display Subsystem

A 3D graphics display card and 21" color monitor provide the ADVISE system with the high performance graphics required to display real-time scientific visualizations. The ADVISE display card is a DEC ZLX-M2, 24-plane Z-buffered system capable of producing smooth-shaded, 3D graphics, anti-aliased lines, texture mapping, and true stereoscopic viewing with appropriate eyewear. This graphics system performs at the rate of 1.4 million 3D vectors per second and 295,000 smooth-shaded triangles per second. The ADVISE display subsystem also incorporates a full-rate video frame grabber to digitize and play back video data for multi-media displays.

3.1.3 Hierarchical Storage Subsystem

The ADVISE storage subsystem consists of 4 GB of traditional winchester hard disk along with over 40 GB of tape storage in a jukebox configuration. Avionics data can be read directly from the tape subsystem or it may be cached to the hard disk for more responsive analysis and visualization. This hierarchy provides ample storage for the raw avionics data streams and for reduced datasets used in typical analysis and visualization operations.

3.1.4 Audio Subsystem

The ADVISE audio system is a Roland SS-55 MkII Sound Canvas MIDI synthesizer containing 128 ROM-based voices, each representing a different orchestral instrument. The synthesizer is controlled by a program running on the ADVISE workstation; the user can assign any desired instrument to a signal and control how the sound attributes change as the corresponding signal changes.

3.1.5 Local Area Network

The ADVISE workstation supports a connection to an ethernet LAN. This network is used to transfer files to other systems on the network, provide a backbone for electronic mail, and enable the ADVISE workstation to share print services from other machines on the network.

Each of the ADVISE hardware components are rack-mounted for easy transportability in the WL/AAAF MERRF van, as shown in Figure 1.

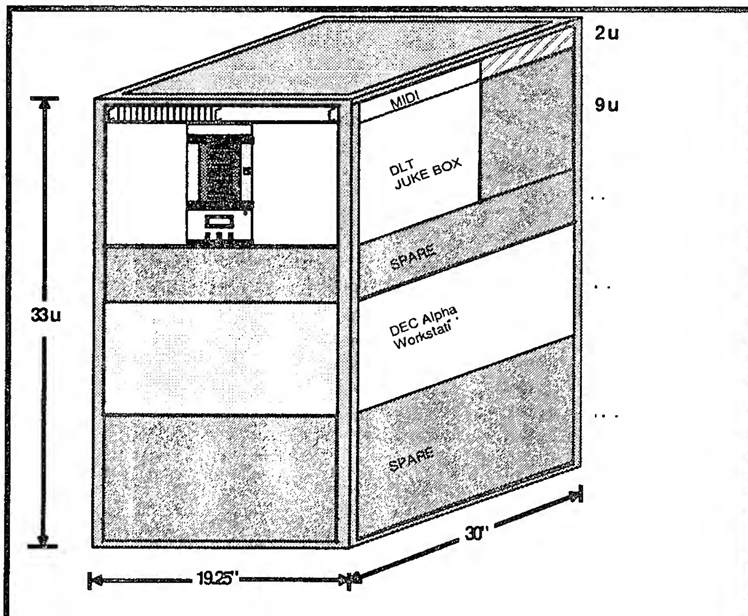


Figure 1 ADVISE Hardware Rack-Mount Configuration

3.2 ADVISE Software

The primary ADVISE software components include a DataBase Management System (DBMS), a data reduction tool, data analysis and visualization tools, an expert system shell, and a hypertext development system. Each of these software components are pictured in Figure 2, and described below.

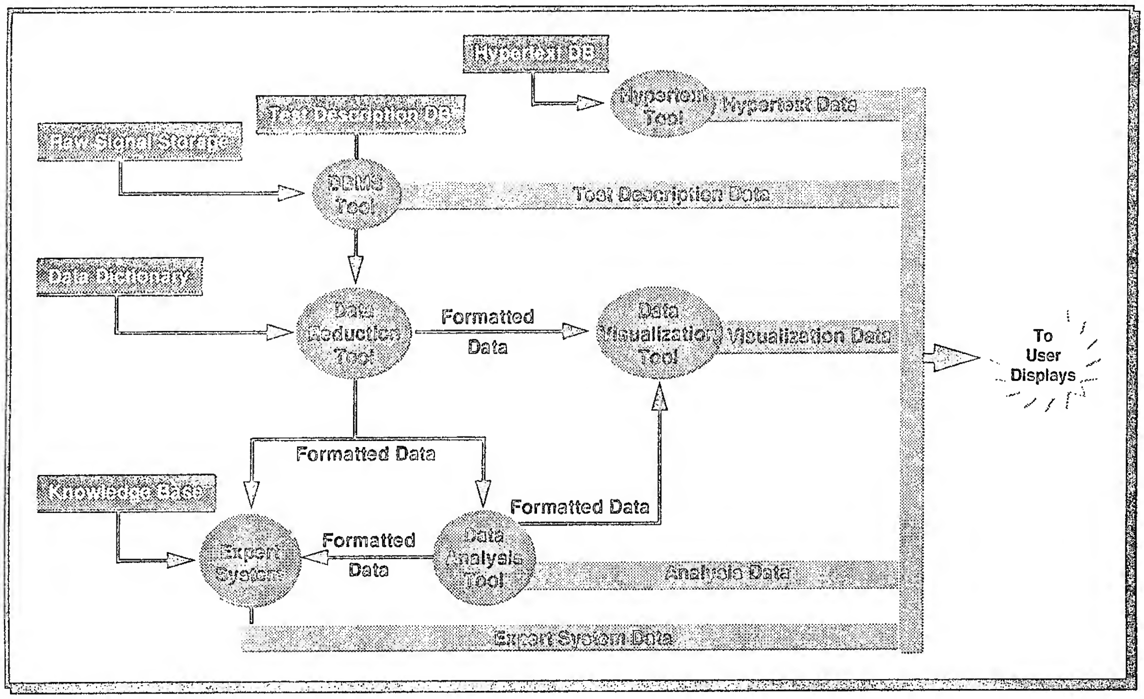


Figure 2 ADVISE Software Dataflow Diagram

3.2.1 Oracle DBMS and the ADVISE-Enhanced EVADE System

The ADVISE database is an extension of the ECM Vulnerability Assessment Data Encyclopedia (EVADE) and is supported by an Oracle DataBase Management System (DBMS). The EVADE database catalogues environmental data about a test, recording such things as where the test took place, who participated in the test, and what general types of information were collected during the test. The ADVISE enhancements to the EVADE system add the capability of storing key extracts from avionics data streams directly in the database along with pointers to the data files containing the full data stream.

3.2.2 Archived Data Reduction System (ADRS)

Before a test or support engineer can effectively analyze a set of avionics data, he must first extract the signals of interest from a typically complicated and convoluted data stream. Signals are difficult to extract from avionics data streams because the avionics system developers are forced to compact a tremendous amount of information into a relatively small bandwidth communications channel. The signals reported on the communications buses also vary with every operating mode of the system, further complicating the task of extracting signals of interest from the data stream.

The ADRS is a data dictionary-based data reduction package that helps alleviate many of the difficulties in accessing data from an avionics data stream. The first step in using the ADRS is to populate the data dictionary. This dictionary defines the format and location of every variable of interest in a data stream. Typically, a single engineer who is familiar with the formatting of data in the data stream uses the ADRS Graphical User Interface (GUI) to enter this format and location information into the data dictionary. Then, all other engineers use a GUI to request any set of signals from the list of signals in the data dictionary, along with a start and stop time. The ADRS software then satisfies these requests by extracting the desired information from a data stream using the data dictionary information previously defined. Each signal is written to a separate data file in a standard External Data Representation (XDR) format for easy transportability between different hardware systems and various analysis software packages.

3.2.3 Precision Visuals PV-Wave Data Analysis Software

For the primary ADVISE data analysis and visualization package, we selected Visual Numerics' PV-Wave, the premier data analysis and visualization package available as a commercial off-the-shelf product. This package provides the ADVISE system with a comprehensive set of analysis functions, a user-friendly interface, and numerous display options. However, one drawback of the PV-Wave software is that it performs its own display coordinate transformations and uses X-windows draw commands for all display operations. This approach ignores the very high performance graphics engine available on the ADVISE workstation and performs all the compute-intensive display calculations on the host CPU, a much less capable processor for this particular application. Thus, in order to support real-time 3D animation displays, ADVISE supplements PV-Wave with a directly callable library of Open3D graphics routines. This combined approach to satisfying ADVISE display requirements relies on the user-friendly PV-Wave package for most displays and provides an Application Programming Interface (API) for the more complex, compute-intensive display functions.

3.2.4 NASA CLIPS Expert System

As mentioned in the key technology discussion above, ADVISE includes an expert system. This expert system advises an analyst on specific data analysis and visualization operations to perform, recommends signals to evaluate, and notifies the user about key or anomalous events it detects in a data stream. The ADVISE expert system shell is the NASA C-Language Inference Processing System (CLIPS). The CLIPS shell holds the required expert system rules in a rule base and uses a forward-chaining inference engine to extract knowledge from these rules based on how the user interacts with the rest of the system. For example, the ADVISE expert system primarily monitors the user's interaction with the ADVISE data reduction tool. As the user extracts signals from the avionics data stream, the ADRS software reports this information to the expert system. These inputs then trigger a flurry of activity in which the expert system makes recommendations to the user of other related signals to extract or specific data analysis and visualization techniques to apply. The expert system also queries the ADVISE database to determine if any key events or anomalies were detected in the signal(s) currently being extracted from the data stream. If they were, this information is also reported to the user.

3.2.5 Internet-Compatible Hypertext Tool

As mentioned in the key technology section, the ADVISE software tool set includes an internet-compatible web browser. This web browser enables a user to access and manipulate hypertext documents coded in the HTML format. In addition, if the ADVISE LAN is connected to the internet, then this tool can also be used to directly access over 2.5 million "pages" of hypertext documents stored on web servers throughout the free world.

4. Test Environments and Generic Test Organizations

Various aspects of the ADVISE system are applicable to many different types of test environments and generic test organizations. The following sections highlight how some of these key technologies can be applied to them.

4.1 Avionics Software Support Facilities

The mission of an avionics software support facility is to maintain and enhance the software in an avionics system. A maintenance problem (defect) must first be re-created from a problem report. The software support engineer must then understand the cause of the problem. After a correction is made the results must be checked to verify that the system still works properly. Software support facilities are also tasked to upgrade avionics systems with additional advanced capabilities. In this case, the goal is to add new features and verify they work as planned while not interfering with previous capabilities.

The ADVISE system would benefit software support engineers in several areas. First, in a maintenance context the ADVISE tools can help an engineer verify that he has indeed recreated a problem. This is often no trivial task with traditional data analysis systems. The data reduction tool makes it much easier for support engineers to extract the data of interest without the long lead times required when working with traditional data reduction support organizations that are often located in batch-oriented computer centers. The scientific visualizations forming an integral part of ADVISE enable support engineers to visualize the operation of a weapon system in such a way that problem areas often become more evident than they would with the traditional data printouts. In addition, the ADVISE expert system can be taught to look for a particular event signifying the existance of the problem of interest, and the expert system can then "tell" the analyst whether or not the problem exists in a data set without the support engineer having to do any significant data analysis at all. Finally, the on-line hypertext user manuals make it easy for software support engineers to understand the operation of their systems and find specific information related to solving the current maintenance problem.

In the enhancement mode of operations, software support engineers can use the ADVISE system's extensive analysis and visualization capabilities to evaluate the impact of any change to the avionics software load. Key data from many different tests can be stored in the ADVISE database, where it is available for performing searches. For example, software support facilities can search the database to see where operational test units are firing most missile shots and determine what part of the launch envelope they should emphasize improving. This data can also be collected from operational excercises and live engagements, such as Operation Desert Storm, and loaded into the ADVISE database, where it is then available for querying. The internal timelines of avionics processes can also be monitored using ADVISE visualization utilities, and expert system rules can be entered to automatically notify software support engineers when busted timelines are detected.

4.2 Developmental Test and Evaluation (DT&E)

The mission of DT&E organizations is to validate that maintenance updates and avionics enhancements perform as advertised before systems are released to the operational commands. The ADVISE database provides a wealth of information on problems that may have been encountered in previous versions of an avionics system. This information can help target a test plan to excercise the system in areas which were problematic in the past, and thus stress the system to the maximum extent in the shortest period of time.

ADVISE multi-media displays can record details on each test and synchronize the playback of this video with actual signal data collected in a test. This capability can help DT&E organizations locate the source of any problems that may exist in a new avionics tape. Similarly, the ADVISE expert system can notify the DT&E engineers if any anomalies relating to known problems in the data collected from a test. This can save an analyst many hours of sifting through reams of test data searching for the anomalies manually, as is often the case today. Data sonification and scientific visualization capabilities in the ADVISE system can work together to audibly notify DT&E engineers when specific events occur in a datastream they are currently visualizing. Validity conditions can also be sonified so that an engineer knows whether a signal he is visualizing on a particular display is valid at any point in time. These tools all help the engineer better understand the data he is tasked with analyzing, and thus make him more efficient in finding problems or in convincing himself that everything is indeed working as expected.

4.3 Operational Test and Evaluation (OT&E)

The mission of an operational test and evaluation facility is to validate that an avionics system works as advertised throughout the life cycle of the system. Fielded units are taken directly off the line and brought into a facility for testing. The tests can include flight tests and live missile launches. Data collected in these tests verify whether or not the systems still work after being stored in active units. Many times, the entire weapon system (including aircraft, missile, flight crew, and even the maintenance team) is evaluated in the process. Two examples of OT&E facilites are provided below: Radar Test Facilities and Missile Test Facilities.

4.3.1 Radar Test Facilities

Radar systems comprise some of the most technologically challenging avionics in a modern aircraft or missile. These systems produce a large quanitity of data on several different high speed buses. For example, the Westinghouse APG-68 radar in an Air Force F-16 fighter produces several megabytes per second of data on a combination of four distinct buses. It is thus difficult to verify whether or not a system is operating correctly from the low level of detail available on these buses. Unfortunately, improper operation sometimes does not show up on the summary displays provided to the pilot or the test engineer. In these cases, the avionics analyst must look at the detailed bus activity to identify improper operating conditions.

The ADVISE system can support the radar engineer in several ways. For example, the storage management features in ADVISE can store complete data sets for detailed analysis while highlighting key extracts in the ADVISE database. The ADVISE expert system can look for specific anomalies that are also recorded in the database. These features can help an analyst locate problem areas in the radar. The ADVISE database is also a storehouse for archiving test conditions that resulted in improper operation on previous versions of an avionics software load, and the hierarchical storage system provides access to the complete data sets collected during these problem tests. ADVISE thus provides excellent examples for regression testing the radar avionics.

The data analysis and visualization capabilities that form the heart of the ADVISE system are invaluable tools in understanding the operation of a complicated radar system. For example, a 3-dimensional animation of a range doppler display can provide dramatic visual evidence of a radar that does not properly counter various types of electronic counter measures. Of course, this tool can also be used to validate the proper operation of the radar in these circumstances.

The ADVISE data reduction tool is especially powerful when it comes to extracting data from a radar bus. The radar buses, more than any other buses in a modern aircraft, contain tightly compacted highly cryptic codes for various signals and flags reported by the various radar Line Replacable Units (LRUs). These buses are difficult to reduce to single-unit files in standard engineering unit formats. However, the ADVISE system includes a powerful data reduction tool that enables an experienced analyst to enter descriptions of each desired data element into a data dictionary one time. The ADVISE data reduction tool can then be used by other analysts that have little or no familiarity with the location and format of a signal in the data stream to extract these signals from the stream into an easy-to use data file that is compatible with most any analysis and visualization tool. This capability enables every engineer in the radar test facility to extract whatever data item he needs by merely knowing the name of the variable. It is no longer necessary for these engineers to submit batch requests to a centralized data reduction facility to gain access to signals they are interested in analyzing. Nor do they need to write special-purpose software to extract their data from the data stream and most engineers are not required to have the intimate knowledge of format and layout that current engineers must be burdened with. They can thus spend their valuable time analyzing the data rather than spending their time trying to find it!

4.3.2 Air-to-Air and Air-to-Ground Missile Test Facilities

OT&E missile test facilities are responsible for verifying that missile seekers, flight control systems, and warhead fuzing functions all operate correctly in fielded missile systems. Before launching a missile at one of these facilities, the warhead is typically removed and replaced with a telemetry (TM) pack. The TM pack interfaces with various signals available within the missile and transmits this information to a ground station for real-time display and post-mission analysis.

During a typical test, a cockpit camera records the Heads-Up-Display (HUD) and/or the radar screen as the pilot maneuvers his aircraft into the correct parameters for a missile launch. After launching the missile, this camera often picks up the exhaust trail from the missile as its rocket motor powers it toward its target. Communications between the pilot and his wingmen and ground support personnel are also recorded.

The first task in analyzing a missile test is to determine if the missile performed correctly. The ADVISE system can support this analysis by displaying any one of several graphics. For example, key parameters from the telemetry data stream can be displayed in electronic strip charts. The ADVISE sonification system can be used to recreate and evaluate the missile tone the pilot should have heard before launching an AIM-9 missile. Three-dimensional animations of range-doppler matrices on radar-guided (e.g. AIM-7) missiles can be displayed. Scoring system data can be displayed and evaluated to determine if the missile closed within a specified range of the target. Fuzing signals can be displayed in synchronization with these other displays to determine the end-game parameters at the time the warhead would have detonated (had it not been removed). The ADVISE data analysis and visualization system can perform all these tasks.

If it is determined that the missile did not perform within established bounds of acceptance, then the missile test facility personnel must determine the cause of the malfunction. Often, this process starts with a review of the mission's cockpit video tape. The ADVISE mult-media displays support this function by digitizing the video and making any frame instantly available for viewing. As the video is played, the ADVISE system can synchronize displays of other signals with the data being displayed in the video, including scrolling strip charts, 2 and 3-D

animations, and sonification "displays". This capability enables the analyst to quickly zoom in on any trouble areas and get a total picture of all the events happening at that point in time.

The ADVISE DBMS and storage management system would enable the analysts to look for previous tests with similar launch conditions to see if the problem is a recurring one. Once a test is located using DBMS queries, actual data from the test can be extracted in real-time using data pointers stored in the database to find the physical location of the tape containing the entire set of data from the previous test. The desired tape can then be loaded into the near-line storage system, or accessed directly if it is one of the tapes already stored in the jukebox.

The ADVISE expert system would prove invaluable to an analyst tasked with identifying any problems that could have caused a missile to fail. The expert system is programmed to automatically recognize anomalous events and report these events to the analyst. This feature can save countless hours of analysis by performing a comprehensive look at all aspects of the test. The analyst may then concentrate on determining which anomalies are expected and which are likely candidates for contributing to a missile fault. The number of anomalous conditions to evaluate will typically be much less (probably less than one signal in a few hundred or a thousand) than the total number of signals recorded in a test, thus saving the analyst a considerable amount of time.

5. Summary

This paper described the key technology areas the ADVISE system applies and extends in the area of avionics systems analysis and scientific visualization: multi-media displays, data analysis expert systems, data sonification, storage management, and hypertext user manuals. The hardware and software components of the ADVISE system were then described. Finally, examples were given of how ADVISE may be applied in various test environments and generic test organizations, including avionics software support facilities, DT&E organizations, and OT&E organizations (such as radar test facilities and missile test facilities). The ADVISE system promises to empower avionics analysts with new tools that will make them more efficient and help them to better understand the systems they are tasked to analyze.

An Overview of the Avionics Data Visualization Integration System Environment's Archived Data Reduction System

Darryl O. Freeman

Science Applications International Corporation
429 S. Tyndall Parkway, Ste H
Panama City, FL 32404

Abstract

This paper describes the Avionics Data Visualization Integration System Environment (ADVISE) Archived Data Reduction System (ADRS), an on-going development effort sponsored by WL/AAAF-2 under the Embedded Computer Systems Readiness (ECSR) contract.

ADVISE provides a powerful computer workstation tailored to meet the needs of avionics test facilities. It provides a prototype capability for the quick and detailed analysis of avionics data from independent and integrated sources using advanced data analysis, visualization, and expert system techniques coupled with state-of-the-art display and workstation hardware. The ADRS is the component of ADVISE which provides the software and user interfaces necessary to extract data from archived avionics data sets and present the extracted data in a format useful for data analysis and visualization.

Introduction

In today's avionics environment, large quantities of data are collected from various data busses. The data volume coupled with high data rates often make it necessary to compress the data into single, packed data streams stored in large data sets. Currently, analysts must research the method in which the data was packed before it can be extracted. Then, they must write specialized software to read the data set and extract the data for analysis. The packing and storage methods used often vary between systems, requiring different data reduction programs for each avionics configuration. To reduce the costs associated with extracting and analyzing these complex data sets, a system is needed which enables an analyst to process large quantities of data efficiently and cost effectively without regard to the method in which the data was packed and stored. The ADVISE ADRS was designed to meet these requirements.

The ADRS is a cost effective environment for data reduction. The ADRS and its Data Dictionary provide an analyst with tools that allow the layout of an avionics bus and its associated signals to be defined once and reused often. The ADRS is written in a manner that allows the user to define multiple data dictionaries without having to rewrite or change software. All data dictionaries are immediately available for use. ADRS allows the analyst to convert large quantities of data into

smaller, more manageable formats which may be used in many state-of-the-art data analysis packages efficiently and with minimum effort.

As shown in Figure 1, the ADRS has three major components: the User Interface, the Data Dictionary Manager, and the Visualization and Analysis Bridge (VAB). The User Interface allows the user access to the various functions of the ADRS. The Data Dictionary allows the ADRS user to create, modify, and store signal definitions in the Data Dictionary. The VAB allows the ADRS user to select signals from the Data Dictionary for data reduction, visualization, and analysis. The VAB parses the archived data set, extracts the data associated with user-selected variables, and creates External Data Representation (XDR) standard output data files for each variable. The output data files can then be loaded into a visualization tool such as PV-Wave for display and analysis. This paper provides a detailed description of each component of the ADRS.

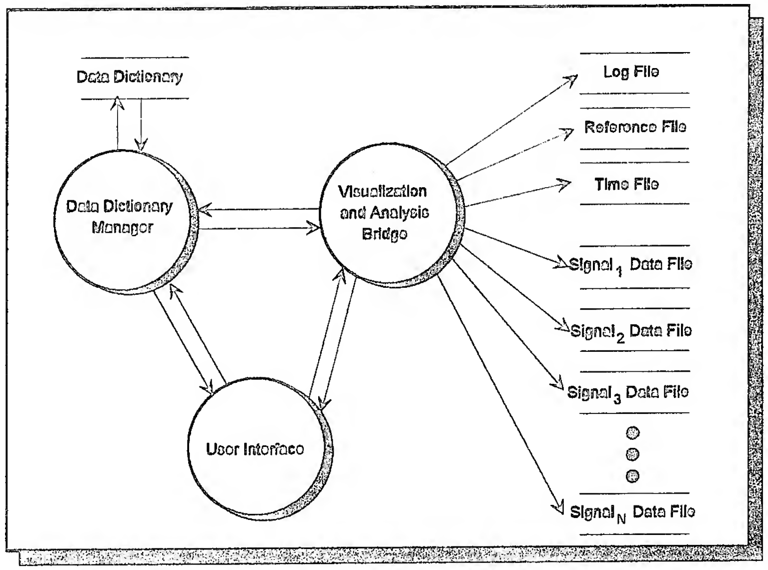


Figure 1 ADRS Data Flow Diagram

User Interface

The ADRS User Interface allows multiple users with little or no knowledge of the underlying signal representations or data packing methods to select signals from a large data set for reduction. One analyst loads the data dictionary and then all users simply select the signals and data set they want to use. The ADRS User Interface provides a user-friendly, industry standard interface to reduce large data sets.

The ADRS User Interface provides an integrated environment for accessing and maintaining the Data Dictionary, selecting signals for data extraction, and initiating the data reduction. The User Interface uses code based on OSF/Motif to provide its industry standard, user-friendly environment. It is comprised of two major components -- the Dictionary Maintenance component and the ADRS Configuration component.

User Interface - Data Dictionary Maintenance Component

The Data Dictionary Maintenance component of the User Interface provides the user with low level access to the ADRS Data Dictionary. It allows the user to add, delete, and update signal definitions,

as well as to query the Data Dictionary for current signal definitions. **Figure 2** shows an example Data Entry Form for adding a signal to the Data Dictionary.

The ADRS Signal Definition Form provides fields which allow the user to load the signal name associated with each signal in the dictionary; comments pertinent to the signal currently being defined; the stream name and sequence number from which the signal is derived; a data type for the signal; the engineering units associated with the signal; the location of the signal in a data buffer including its offset into the buffer, its start bit at that location and its length in bits; and the scale

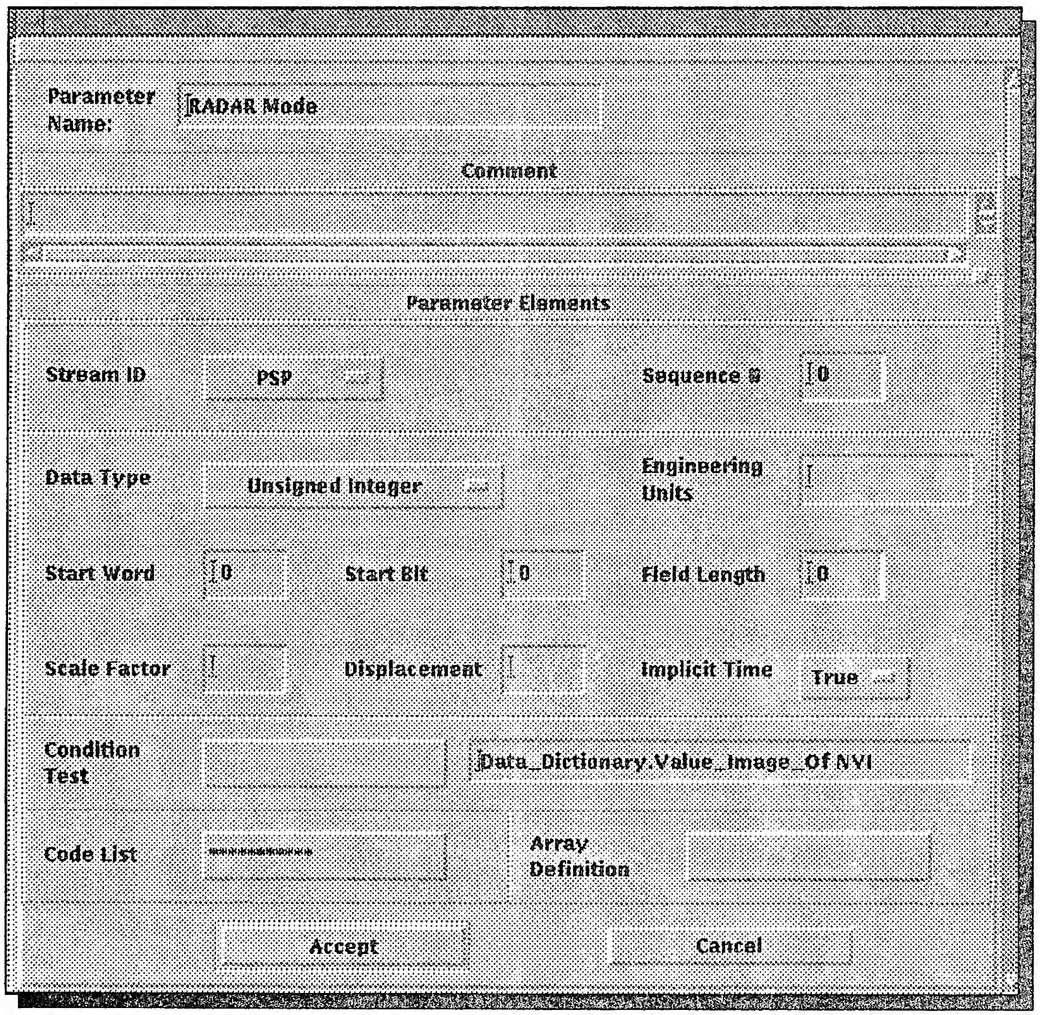


Figure 2 ADRS Signal Definition Form

factor associated with the buffer. Additionally, the ADRS Signal Definition Form allows the user to define the list of possible values associated with signals that can be enumerated. And finally, the ADRS Signal Definition Form allows the user to define the array elements for signal types that are expressed as arrays.

User Interface - ADRS Configuration Component

The ADRS Configuration component of the User Interface allows the user to enter configuration parameters necessary to reduce the data set. These configuration items include the name of the data dictionary, the start and stop times for the data extraction, the signals for which data will be extracted and output filenames for status and reference information.

When initiating the data reduction process, the ADRS Configuration Form provides a field for the user to enter the name of the data dictionary. The ADRS software reads the selected dictionary and extracts all signal names stored in it. The ADRS Signal Selection Form presents the list of signals to the user in a traversable format that allows each signal to be marked or not marked for extraction as desired by the user. When the signals of interest have been marked, the user is allowed to store the signal list for use in later reductions. The list of marked signals is passed to the Visualization and Analysis Bridge where it is used to process the input data file and extract data associated with the selected signals.

The ADRS Configuration component also allows the user to enter the time interval for which data will be extracted. The time fields allow the start interval and stop interval to be specified with millisecond accuracy. To support total reduction of the data set, the start time interval defaults to the first start time in the data file while the end time interval is set to the last day, hour, minute, and second of the year.

Finally, the ADRS Configuration component allows the user to change the default output filenames that will be used to store status information for the extraction process, time stamp information for each data block, and signal cross reference information for all selected signals. These output files are discussed later.

Data Dictionary Manager

As stated previously, in current avionics environments users must know low-level implementation details concerning the data set they want to reduce. With ADRS, only one analyst is required to load specific details about the location and types of signals in a data set. This information is then stored in the ADRS Data Dictionary where it can be retrieved by any user with ease. By accessing the data dictionary, any user gains immediate access to all variables in the data set without having to know intimate details about the underlying representation and stream location of the data. Also, once the data dictionary has been loaded, no additional code needs to be written to access the data set. The ADRS provides a standard interface to all ADRS data dictionaries. Any analyst using it is able to reduce a data set without a priori knowledge of the packing and storing methods used for the data set.

The ADRS Data Dictionary Manager provides the low level software maintenance functions for creating the ADRS Data Dictionary. With it, the user can add, delete, and modify dictionary entries. It also provides low level query functions for retrieving stored dictionary entries.

Signal Definition

As discussed previously, the Data Dictionary contains a complete list of signals defined for a system. For the ADRS, a signal is any data element stored in an archived avionics data set. Since multiple signals may be packed into a single word, the ADRS allows the user to specify the start bit and bit length for a signal. Additionally, by using its conditional test capability with its and/or chaining support, the ADRS allows signals to be defined based on the presence and value of other signals in the current data block. Up to 256 levels of and/or chaining in a signal definition is supported by the ADRS Data Dictionary.

The ADRS makes it easy to define signals that may have a limited range of possible values. These enumerated types have a separate definition facility which makes the enumeration type visible to all signals being defined in the dictionary. Therefore, the user is required to enter the definition once, and then it is available for reuse as often as is required. These enumeration types, once defined, are automatically visible to all other signals in a dictionary. Enumerated types are further discussed in a later section.

Finally, the ADRS has a mechanism which allows arrays to be defined. As with enumerated types, once an array type is defined, it becomes immediately available for use by other signals which may have a similar array definition.

Conditional Test Entry

In a many avionics systems, signal location definitions are not constant. Their location and value depend on other signals being present in the buffer. For example, the target signal to noise ratio signal appears only in PSP data busses when the mode is Track TM1 Short. Therefore, a conditional test must be applied to the signal definition. The ADRS solves this problem with the Conditional Test Form.

The Conditional Test Entry Form allows the user to develop simple conditional tests which can be applied to determine the presence of a signal in a data buffer. These conditional tests include the comparative functions <, >, =, and /=; the logical functions * and +; and the boolean functions true and false. **Figure 3** shows the Conditional Test Entry Form.

The form allows the user to enter the name of the condition test, comments specific to the condition, and the condition itself. The left and right push buttons are context sensitive to the condition. The buttons allow the user access to previously defined conditions as well as constants and enumerated types. This allows the user to build complicated tests from previously defined building blocks.

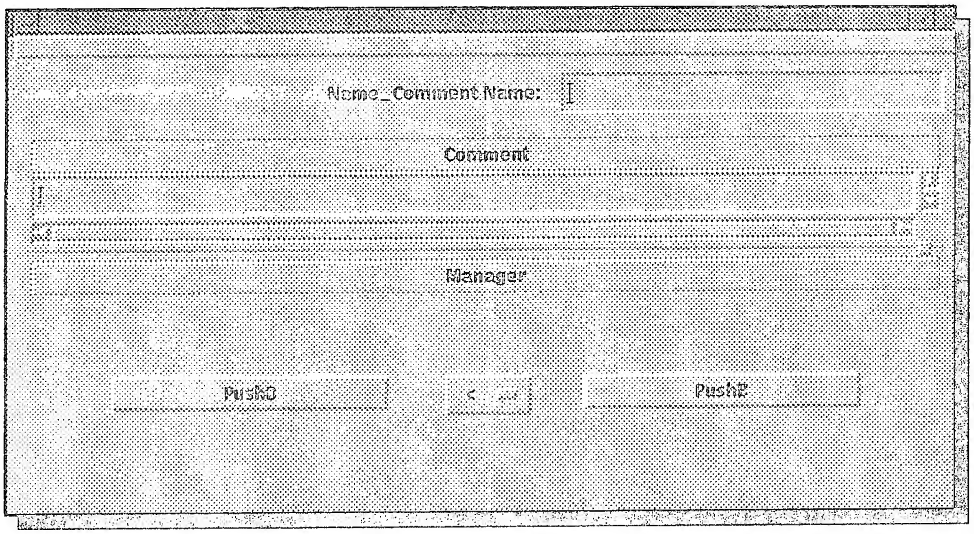


Figure 3 ADRS Conditional Test Entry Form

And/Or Chaining Support

And/or chaining support allows complex conditional statements to be applied to a signal definition. This functionality allows several conditional tests to be combined when defining a signal. The ADRS supports up to 256 levels of and/or chaining in a signal definition. When defining a chain, the data dictionary administrator creates a condition variable for each element of the chain. For example, the chain (A*B)+(C*D) would require three conditional variables to be entered into the dictionary: V1 = A*B, V2 = C*D, and V3 = V1+V2. This mechanism allows maximum flexibility and reuse of data dictionary entries since each component of the chain is separately stored in the data dictionary. Therefore, all and-conditions and or-conditions may be reused by other and/or chains in the dictionary. Also note, the individual conditions of the chain may themselves be complex statements. In the example above, condition A may have been previously entered into the dictionary as a simple statement like VarA < Constant or it may be a series of complex and/or conditions. By using this method, ADRS ensures that a variable may be completely defined with minimum effort.

Enumeration Types

The ADRS has built in facilities to aid the user when defining variables which have more commonly known values. For example, a typical avionics system may have modes Range While Search, Track TM1 Short, and Track While Scan. Rather than referring to these modes by their numerical representations, the ADRS allows these mode names and their values to be defined in a common variable called Mode. Then the more common name may be used in the definitions.

Signals which have a finite range of possible values are defined using enumerated types. Since more than one signal may share the enumeration, the ADRS Data Dictionary allows an enumerated type

to be defined separately. Signals can then use the type by referencing the enumerated type's definition in the Code List field of the ADRS Signal Definition Form (see **Figure 2**).

The ADRS Enumerated Type Entry Form, shown in **Figure 4**, is used to define enumerated types. This Form allows the user to enter the name of the enumerated type as well as the name and value of each element of the signal. For example, an enumerated signal Mode_Type may be defined to have the modes Range While Search, Track TM1 Short, and Track While Search with corresponding values 8, 18, and 22.

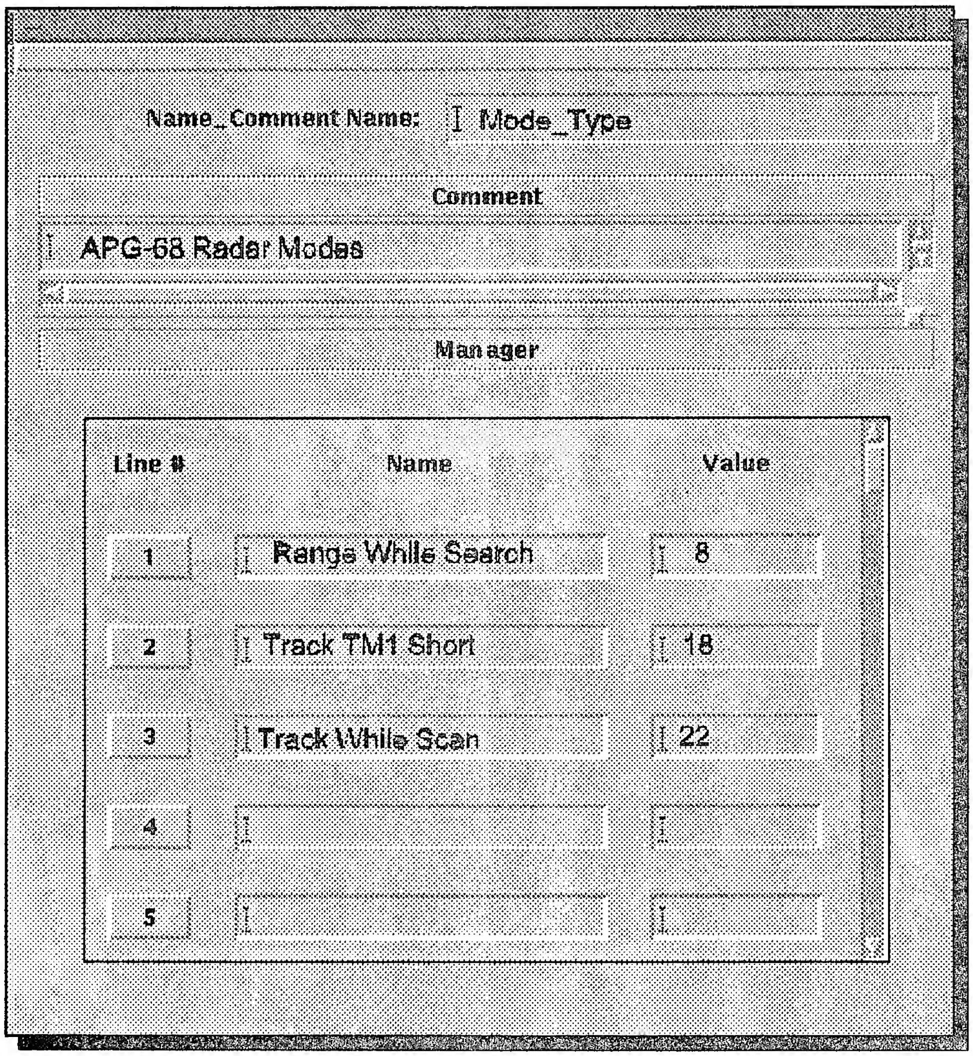


Figure 4 ADRS Enumerated Type Entry Form

Array Support

The ADRS Array Definition Form is used to define array signals. The form allows the user to specify the size, elements, and element locations of array signals. Arrays of signals can be stored in the dictionary and referenced by other signals whose elements are similar to the array definition.

Visualization and Analysis Bridge (VAB)

The ADRS Visualization and Analysis Bridge provides the functionality necessary to extract data from the archived mission data set. After selecting the signals for extraction, the VAB is called with the signal list, the time interval, and the data set. To keep each reduction run separate, the VAB creates a directory based on the time interval to store the reduced data sets. The VAB opens the data file and parses the requested data set, extracting the data for each user-selected signal. The extracted data is stored in a data file whose name matches the signal name. To maximize compatibility and portability, the data in the ADRS output files are stored in XDR format.

The ADRS VAB creates other files to assist the user in reading the output data files. These include the Log File, the Reference File, and the Time Stamp file.

Visualization and Analysis Bridge Log File

The ADRS creates a log file which contains information about the extraction process. The definition for each signal selected is logged to the file as are the output filenames, the list of extracted signals, and all error messages that were displayed during the extraction process. The log file also contains status messages indicating the progress of the reduction. The log files are numbered sequentially and appear in the local directory. The log file's default filename is *Extract_Archived_Data.Log.#* where the # represents the number assigned to the current log file. However, this name may be changed by the user in the initialization process.

Visualization and Analysis Bridge Reference File

The reference file is a text file which contains significant information about each signal extracted. It is created to aid the visualization tool to bridge the data file and the Time Stamp file. It contains the following information:

Signal Name: The signal name.

Data Type: The XDR equivalent data type of the signal's data.

Time Filename: Filename for the file containing the time stamps for each buffer of data.

Buffer Size: The number of data samples for the signal in each buffer.

Sample Rate: The rate at which the data samples were created.

Scale:	Currently unused and set to 1.0.
Offset:	Currently unused and set to 0.

Visualization and Analysis Bridge Time Stamp File

The data set contains a time stamp for each buffer of data. This time stamp is extracted from the buffer and stored in an output file named *Time_Stream_Name* where *Stream_Name* is replaced with the name of each stream in the data set. One time stamp file is created for each stream. The data in the file is recorded from the time stored in the header of each data buffer. Information in the reference file is used to interpret these times when reconstructing the data and its time line.

Visualization and Analysis Bridge Signal Output File

The ADRS creates one output data file for each signal selected for extraction. The extraction files are located in a local directory whose name is created from the user-selected start time. The data files are stored in this directory with names matching the signal name. All data in the output files are in External Data Representation (XDR) format. This standard allows maximum system portability of the data. The output data files, the time stamp file, and the reference file allow the analyst to easily import the reduced data set into analysis and visualization tools.

Summary

This paper has described the ADVISE Archived Data Reduction System. The ADRS is a very flexible, and easily extended avionics data reduction tool set which aids the analyst in reducing large quantities of interleaved data into smaller, non-interleaved data sets. These smaller data sets are ideal candidates for advanced visualization and analysis. The ADRS outputs all data in XDR format thereby ensuring the portability of the data. Its Motif-based User Interface ensures its standardization and ease of use. The novel methods used to create and maintain the ADRS Data Dictionary also add to its ease of maintenance and make the ADRS a valuable tool for analysts inside and outside the avionics arena.

Artificial Neural Networks for Real Time Verification of Ballistic Chamber Pressure Data

Jeffrey S. Murter
Instrument Development Division
U.S. Army Combat Systems Test Activity
Aberdeen Proving Ground, MD 21005-5059, USA

Gary C. Fleming, Francis L. Buck, Katherine Y. Ernhart, Christopher D. Hash
SFA, Inc.
Landover, MD 20785-5322, USA

ABSTRACT

The U.S. Army Combat Systems Test Activity (CSTA) currently examines ballistic chamber pressure data records for validity of transducer installation and functioning. These examinations occur on-site immediately after acquisition. Real-time decisions are then made regarding whether or not testing should proceed or whether transducer problems require correction. The quality of the decision currently depends upon the knowledge and experience of the test technician operating the data acquisition system. Access to an expert may be required to ensure correct decisions are consistently made for high priority or problematic tests. CSTA has undertaken a project by which the chamber pressure data records are verificed automatically, in real time as the data are collected. A layered, feedforward neural network has been developed to model the pressure response of live-fire testing. The network is rapidly trained in real time using conjugate-gradient descent. Parameters from the trained network are used to perform a preliminary analysis of the data. The network is retrained four additional times on smaller time intervals to support more detailed analyses relevant to known chamber pressure processes. These analyses are implemented on derived data sets formed from the residuals of the neural-network model and the pressure-chamber data. The process has been shown to be effective for detecting electrical connection faults, blowby, and ringing. It is currently be implemented as part of the live-fire data-collection software system.

1. INTRODUCTION

The U. S. Army Combat Systems Test Activity (CSTA) has a need to evaluate ballistic chamber pressure data records on-site as weapons testing is being conducted. Measurements of the weapon chamber pressure during firing is critical for determining both the safety and combat effectiveness of a weapon. The CSTA currently relies upon expert diagnostic personnel to be present or on-call during ballistic weapons testing operations. An automated system that effectively subsumes this human expertise will free these personnel for other tasks, allow CSTA to schedule weapons testing independently of the schedules of these expert personnel, and save money.

Weapon chamber pressure is measured after a set of electrical pressure transducers is placed in various positions of the gun tube. Accurate chamber pressure measurements depend upon the installation of the individual transducers, proper operation of the transducers, proper electrical

connections, and proper signal handling. The collected data for a single transducer during a firing test yields a profile of the chamber pressure as a function of time. This trace is called a P-T curve. Data are typically collected at a rate of 200 KHz (20,000 data points in a 100 millisecond interval). If a transducer is properly installed and functioning, the measured P-T curve is relatively smooth and shows a rapid rise in chamber pressure with a somewhat slower fall-off. A typical P-T trace is shown in Figure 1.

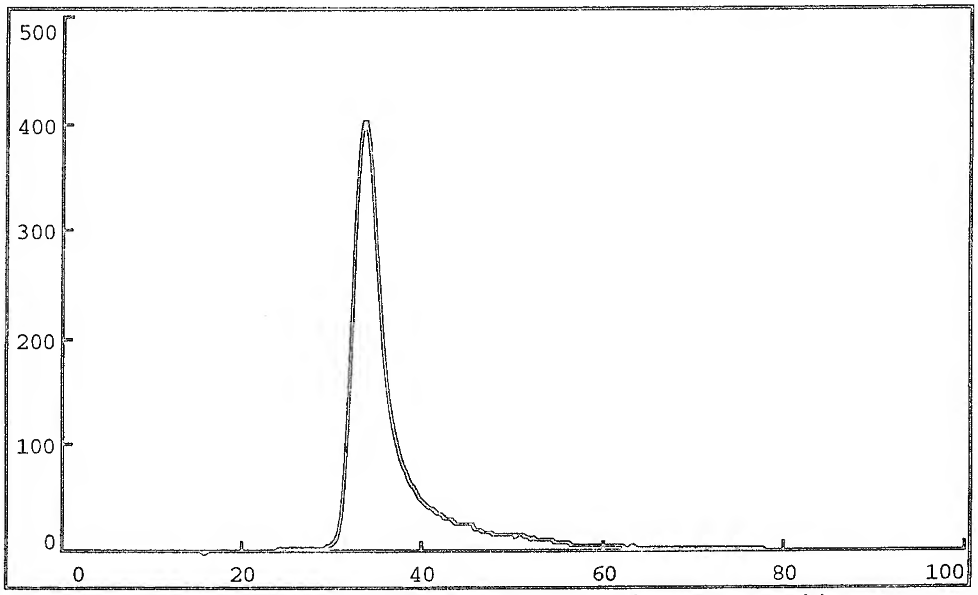


Figure 1 - A typical, normal P-T trace. Time is measured in milliseconds, pressure is measured in megaPascals.

Deviations from this general profile are used to diagnose problems in the installation or functioning of individual transducers. Such problems in general are classified as improper grounding, electrical drift (lack of insulation), blockage, resonance or ringing, baseline offset, blowby, thermal transient effects, and discontinuous electrical connections. Each classification can be associated with a corrective action to be applied before further weapons testing is resumed.

Connector faults arise when an electrical connection is not securely made so that the resulting circuit is only made intermittently during ballistic chamber pressure (BCP) testing. As shown in Figure 2, the resulting P-T curve is characterized by a general blockiness and often a failure of the pressure measurement to return to 0 megaPascals.

Blowby occurs when the mounted transducer has a leaky seal resulting in escaping gas. Note that the curve shown if Figure 3 shows no abnormalities. However, Figure 4, which is the same data viewed at a different scale shows a characteristic deformation in the top portion of the P-T curve. This shows that an enlarged scale is required to notice the irregularity. We refer to this change of scale as focus of attention or FOA. A major thrust of this effort describes how the FOA can be defined and manipulated by an automated system.

The phenomenon of ringing occurs when a resonance occurs within the volume around the transducer. Transducers are not mounted flush with the interior wall of the gun barrel; this space is packed with an incompressible grease. If not enough grease is used, an organ-pipe effect may result with resonant frequency of approximately 12 KHz. As shown in Figure 5, it is characterized

by a high frequency response in the fall-off portion of the P-T curve. As with blowby, human operators often require a data presentation at a diffferent scaling to observe ringing artifacts.

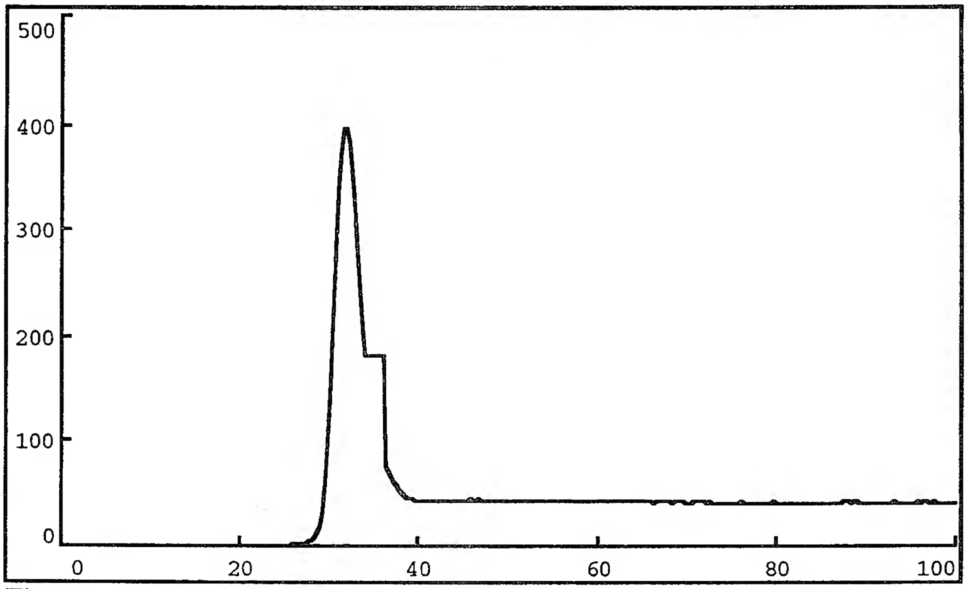


Figure 2 - An example of a P-T data trace resulting from a faulty electrical connector. Note the jump discontinuities and the failure of the pressure values to return to 0 MPa.

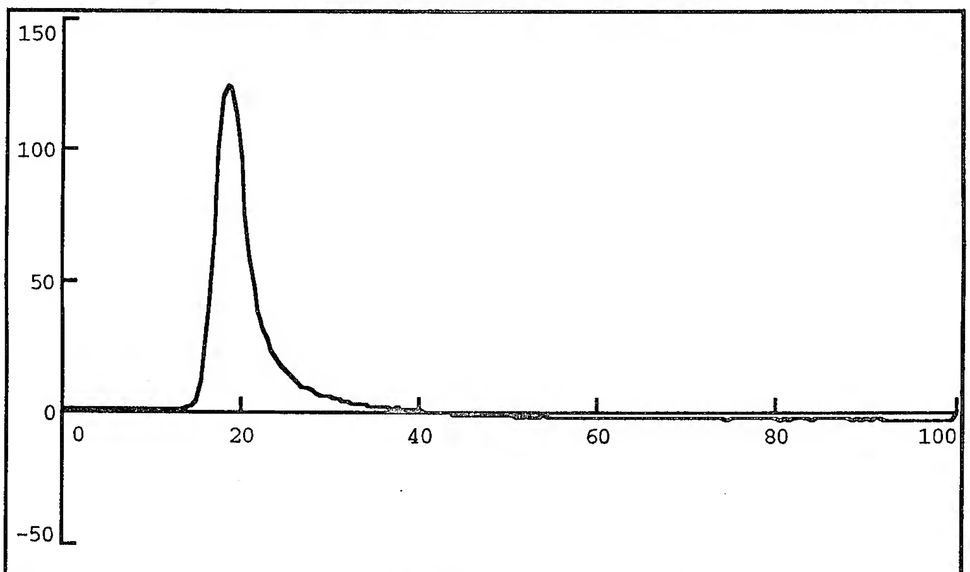


Figure 3 - An example of blowby. Note that it is not possible to detect the defect in the top of the curve at this scaling.

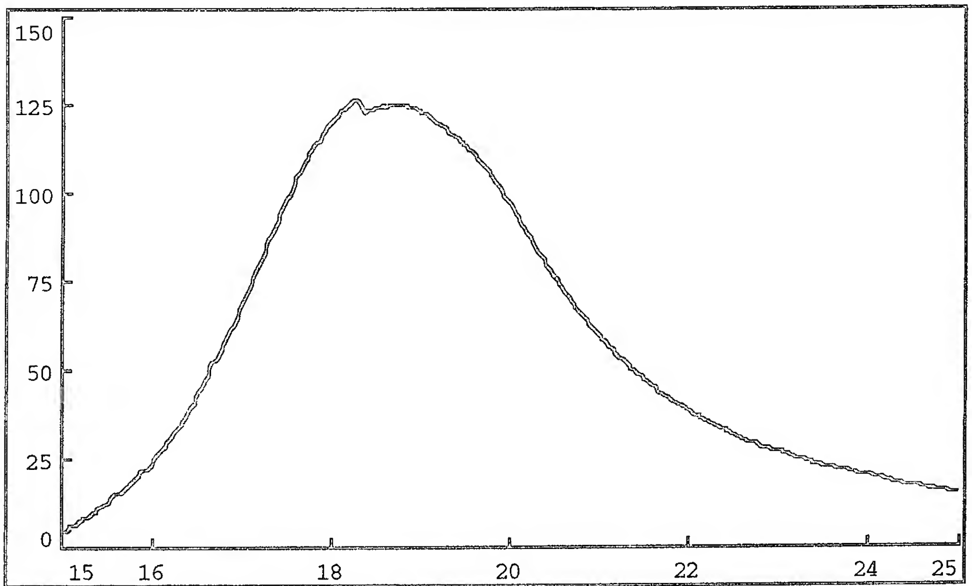


Figure 4 - Enlargement showing the blowby artifact. The only characteristic of blowby displayed by the P-T trace is the "cleft" in the uppermost portion of the graph.

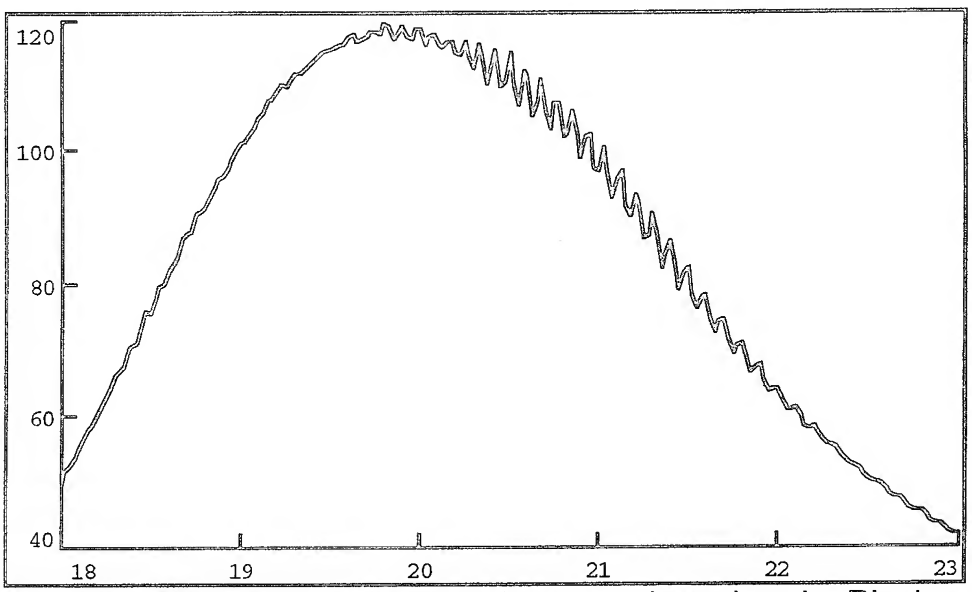


Figure 5 - Ringing artifact displyed on an enlarged scale. Ringing appears as a resonance beginning slightly before the pressure maximum and damping out after approximately 3 milliseconds.

2. EMBEDDING A MODEL IN A NEURAL NETWORK

A human expert classifies P-T curve anomalies by comparing the data derived P-T curve with an expectation of an idealized or general P-T curve model. While a human handles this cognitive task with ease, an automated AI-based system must match an internal general representation to the test data. This is a registration problem. The idealized model must be shifted and elastically deformed so as to match the data as closely as possible. This task is equivalent to selecting values of morphological parameters to remap the general P-T model. For example, suppose the function $p = f(t)$ describes a general P-T curve where t is measured in milliseconds and p is measured in megaPascals. This model may be modified with four fundamental transformations. These transformations are vertical translation (pressure offset component), vertical scaling (amplitude), horizontal (temporal) translation (time of onset), and horizontal (temporal) scaling (or weapons system response). Thus, if f is a model of the general pressure-time response, a particular data set can be modeled by the following equation:

$$p = p_0 + Af(\omega(t - t_0)) \tag{1}$$

The parameter t_0 denotes a temporal translation and has units of milliseconds, ω denotes a temporal scaling (contraction or dilation) and is unitless, A denotes an amplitude scaling and is unitless (as f has units of megaPascals), and p_0 denotes a vertical translation and has units of megaPascals. The parameters ω and t_0 are "interior" to the function f, and thus their determination is a problem in nonlinear regression. Preprocessing can be used to derive initial values of the parameters. For example, the location of the maximum chamber pressure can be used to estimate A and t_0. The 3 dB falloff points can be used to estimate ω. Assuming that f is differentiable, the technique of gradient descent can be used to fine-tune the estimated values of the model parameters.

A layered, feed-forward neural network can be used to embed the P-T curve model. The model parameters are represented as network weights and the nodal transfer functions implement the P-T model. Such a network is shown in Figure 6. A gradient descent technique (such as back-error propagation or conjugate gradient descent) may be used to fine-tune the parameter values. This geometrically-based implementation has a distinct maintenance benefit: if the general P-T model changes or evolves with experience, then modifying the registration portion of the anomaly detector reduces to updating the feedforward network topology and (possibly) redesigning the initial parameter estimation techniques. The gradient descent algorithm need not be modified for the revised model.

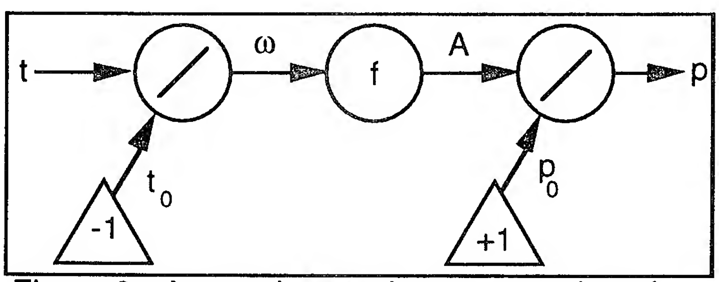


Figure 6 - A neural network representation of Equation 1. Explanations for the icons are given in the text.

The icons in Figure 6 represent nodes with specific transfer functions. Circles represent neurons that accept input values. The transfer function of the neurons with the linear "slash" is the identity

function $g(x) = x$. The transfer function of the neuron labeled f is the generic P-T model. Triangles represent neurons that accept no inputs but continuously output a constant bias value; each such neuron is labeled with that value. The connections between neurons are weighted. The output of a neuron is multiplied by a connection weight before being input to the next neuron. Neurons with multiple inputs sum the weighed outputs of the "upstream" neurons to compute a net input. This value is then modified by the neuron transfer function and is passed "downstream" to other neurons, or as an exogenous output value.

3. THE EMBEDDED PRESSURE-TIME MODEL

We found that the gross characteristics of a P-T sample can be effectively modeled by fitting the data with a generic model that is the difference of two sigmoid (or logistic) functions. The model used is given by

$$p(t) = p_0 + \frac{A_1}{1 + exp(-\omega_1(t-t_1))} + \frac{A_2}{1 + exp(-\omega_2(t-t_2))} \tag{2}$$

This model has three components, an initial pressure value p_0, a sigmoid to model the initial rise of the P-T curve, and a sigmoid to model the fall-off portion of the P-T curve. A graph of a single logistic sigmoid is shown in Figure 7.

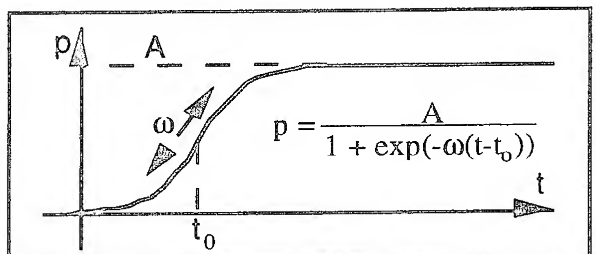


Figure 7 - A generalized sigmoid function produced from a modified logistic function..

The two sets of parameters, A_1, ω_1, t_1, and A_2, ω_2, and t_2 serve the same role (and have the same unit values) as their similarly named counterparts described in Equation 1. The t_i parameter translates a sigmoid, the ω_i parameter controls the temporal scaling (or pitch) of the sigmoid, and the A_i denotes the amplitude scaling - in this case, the right horizontal asymptote of the sigmoid.

The neural network we employ is built from two subnetworks each of which represent a generalized logistic function shown in Figure 8. The generalized logistic subnetwork is shown in Figure 9. The final network used to generate each registration is shown in Figure 10. The parameters A_1, ω_1, and t_1 are represented by the first logistic subnetwork and the parameters A_2, ω_2, and t_2 are represented by the second logistic subnetwork. The parameter p_0 is combined with the outputs of the logistic subnetworks to produce the P-T model value.

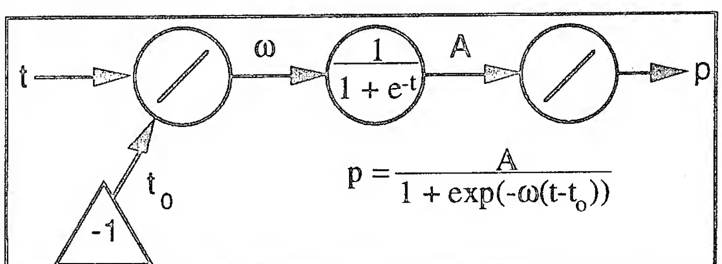


Figure 8 - A generalized logistic subnetwork that implements the function shown in Figure 8.

The seven model parameters, p_0, A_1, ω_1, t_1, A_2, ω_2, and t_2, represented by neural network connection weights, are tuned through the method of conjugate gradient descent. It was hypothesized that this model would account for at least 90% of the variance of a normal P-T data set when

measured on the interval [0 ms, 100 ms] at a sampling rate of 100 Hz, or 1/200th the maximum data rate. This is a reduced data set consisting of 100 P-T points.

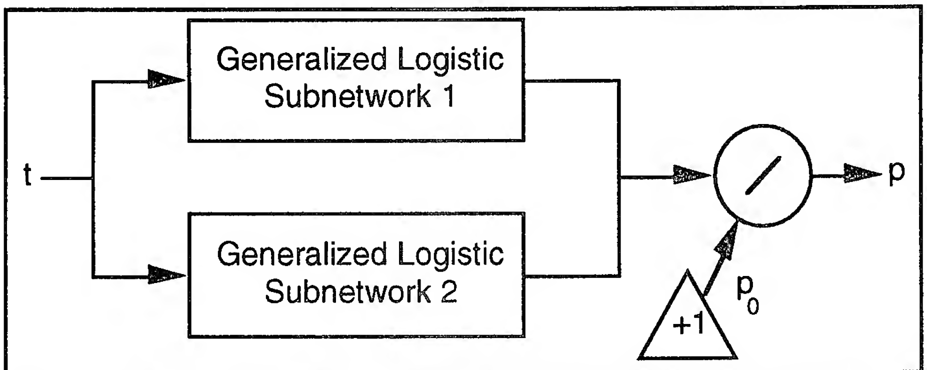


Figure 9 - Neural network representation of Equation 2, the P-T model.

It was found that significant portions of the data variance could be accounted by the seven-parameter difference-of-sigmoids model. The amount of variance accounted by this model on ten data sets is shown in Table 1. In all cases, at least 95% of the data variance was accounted by the network-embedded model. A typical model fit of a normal P-T data set (file A_1) is shown in Figure 10. Of particular interest is the success in modeling the loose connection data curves. One characteristic of these curves is a failure to return to zero pressure after 100 milliseconds. An example of the data fit is shown in Figure 11.

File Name	Fault Type	Variance Accounted
A_1	Normal	95.8%
A_2	Normal	95.8%
T127	Normal	96.3%
R07C5-C	Blowby	96.8%
R08C5-C	Blowby	96.9%
B_1	Connector	97.5%
C_1	Connector	98.8%
T122_1	Connector	97.4%
R04C2-C	Ringing	97.6%
R08C2-C	Ringing	97.7%

Table 1 - Variance accounted by seven-parameter difference-of-sigmoids model on 100 millisecond analysis interval.

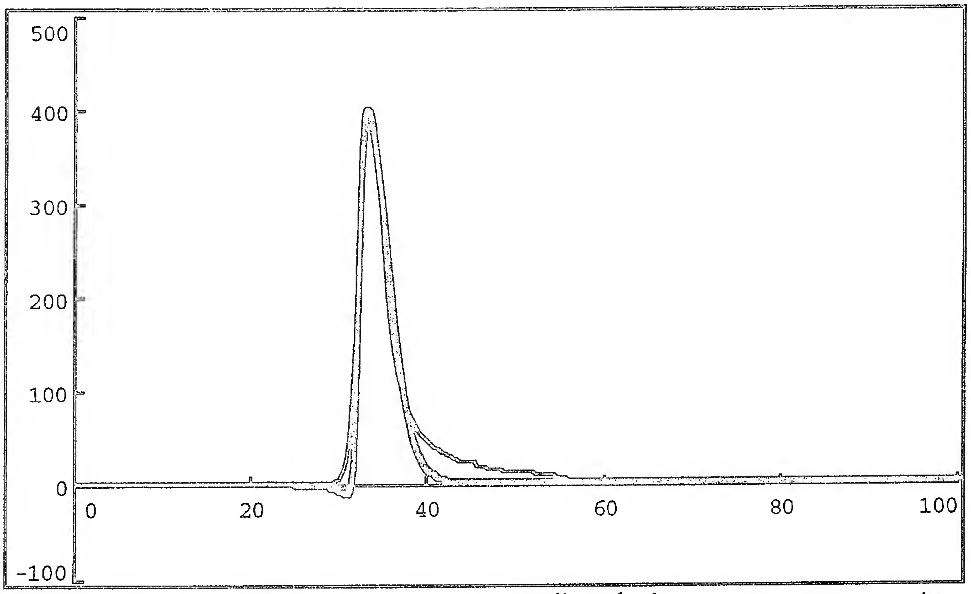


Figure 10 - A least-mean-squares fit of the seven-parameter difference-of-sigmoids model to a normal P-T data set. This fit (heavy line) used 51 data points and seven iterations of the conjugate gradient descent algorithm.

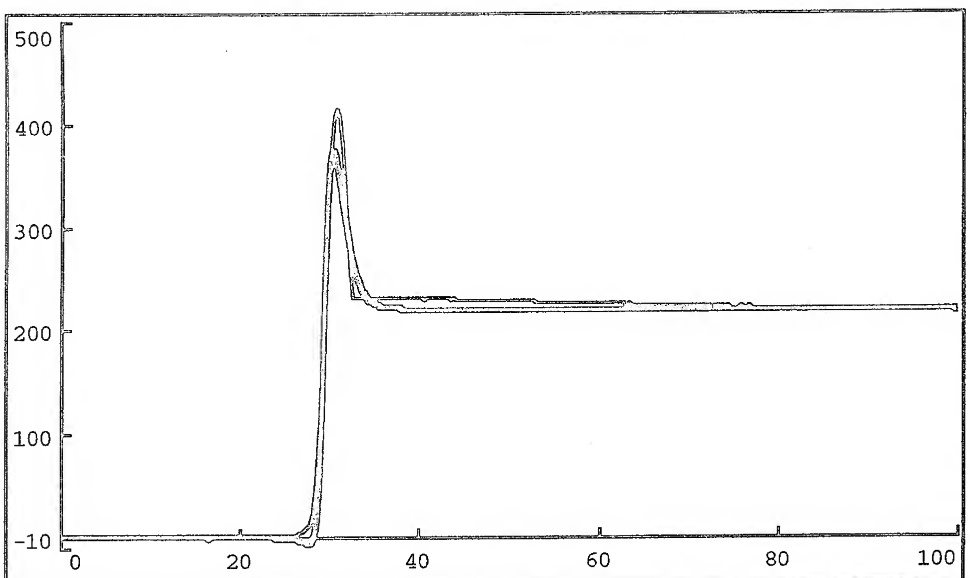


Figure 11 - The seven-parameter model (heavy line) is capable of modeling disparate asymptotes. Inspection of the model asymptote parameters is an indicator of existence of particular fault types. This particular graph displays a loose connector fault.

4. FOCUS-OF-ATTENTION REGIONS

After the registration problem has been solved, the measured data must be compared with the matched model. Each transducer anomaly type is associated with a corresponding data-model mismatch anomaly. A pattern recognition device associated with a particular transducer anomaly is used to inspect the difference between the data and the matched model and to determine if these differences meet the expected differences associated with the corresponding graphical anomaly. Such pattern recognition devices are custom designed for a particular anomaly type and have two components: a set of feature detectors and a feature-set evaluator.

A set of feature detectors for a particular anomaly type is most likely to be fashioned from traditional signal processing techniques. For example, 60 Hz ground loop noise is best detected by passing the data-model difference through a 60 Hz matched filter. More than one procedure may be required to detect the presence of a graphical anomaly. Thus, the output of a set of feature detectors may be considered to be a vector of values with each value being the output of a particular feature detector algorithm.

This feature vector is then processed by a corresponding feature-set evaluator. The evaluator effectively partitions the feature space into classification categories. Each evaluator considers several classification categories, including no evidence for the anomaly, some evidence for the anomaly, and strong evidence for the anomaly.

This modular decomposition of feature detection and classification is inspired by parallel processing models of cognitive processes. It also lends itself to effective maintenance strategies. It is possible to tinker and revise a particular anomaly detection and classification module without disrupting the processing capabilities of the remaining software system.

Finally, the outputs of each anomaly-specific interpreter are integrated to produce an overall P-T curve diagnosis. Because multiple transducer problems can exist simultaneously, the integrator will use cooperative rather than competitive architectures. If no anomalies are detected, then the weapons test operator can be advised to proceed with testing. If anomalies are detected (i.e. blowby, ringing or faulty electrical connection), a list can be displayed. Further, a knowledge-based system can be integrated with the anomaly detection and classification system to recommend specific corrective actions and additional test procedures, as appropriate to the anomaly type.

However, the model fit to the entire data set is, in general, not sufficient for the specific anomaly detectors to properly function. Most of the anomalies are manifest in the ascending or descending portions of the P-T curve. thus, it makes sense to optimize the model fit in this region before engaging the specific anomaly detectors.

Whole-curve registration is a critical first-step in this process. If the data are well behaved, it is relatively easy to automate the location of the critical portions of the P-T trace. But then, if the data are well behaved, there is little need to engage the anomaly detectors. Whole-curve registration allows an automated system to identify critical portions of the P-T trace when the data is not well-behaved. After registration, model parameters and model behavior are used to estimate the location-in-time of the critical portions of the P-T trace.

From these values, four subregions, or foci-of-attention, are defined. These are:

A. The rise-and-fall portion of the curve. This focuses on approximately the 90% of the uppermost portion of the curve.

B. The top-of-curve. This focuses on the upper 50% of the curve, or to within 3 dB of the model peak.

C. The ascending limb. This is the interval that includes approximately 90% of the ascending portion of the curve, terminating at the curve peak.

D. The descending limb. This is the interval that includes approximately 90% of the descending portion of the curve, beginning at the curve peak.

It was hypothesized that difference-of-logistics model would account for at least 95% of the variance of a normal P-T data set when measured on various FOAs at a sampling rate of 200 KHz, the maximal data rate, and the model parameters are readjusted for each FOA.

It should be emphasized that five separate neural networks are represented by these hypotheses: one for the general data fit, and one each for the four foci-of-attention. These five networks are topologically equivalent and represent the same P-T model. What differs is the data set on which the seven free parameters are optimized. This is the salient idea behind the differing foci of attention: optimizing the model fit on a smaller data interval will allow the detection and classification of fault artifacts.

The five different model fits required approximately 20 seconds per channel of computer time on a Hewlett-Packard 750 utilizing a Risc-based architecture and operating at 50 MHz. Almost all of this time was expended in evaluating the mean-square-error function for various values of the seven model parameters. It is believed the conjugate gradient descent algorithm reduces the number of error evaluations by a factor of 10 (personal conjecture) when compared with the backerror propagation technique.

Table 2 shows the variance accounted for each of the four foci-of-attention described above.

		Variance Accounted			
File	Fault	R. & F.	T. C.	A. L.	D. L.
A_1	Normal	99.7	99.5	99.9	97.1
A_2	Normal	99.8	99.4	99.9	98.0
T127_2	Normal	97.2	97.7	99.9	95.9
R07C5-C	Blowby	99.3	99.7	99.8	98.2
R08C5-C	Blowby	99.1	99.5	99.6	98.4
B_1	Connector	98.1	99.7	99.9	86.2
C_1	Connector	99.5	99.5	99.9	99.2
T122_2	Connector	92.2	87.4	99.2	96.6
R04C2-C	Ringing	99.3	99.3	99.9	98.3
R08C2-C	Ringing	99.6	99.2	99.8	99.8

Table 2 - Variance accounted for each of the four foci-of-attention: Rise and Fall, Top of Curve, Ascending Limb, and Descending Limb.

In all but two cases the model was able to acount for at least 90% of the data variance. Those cases in which less of the variance was accounted is due to the gross deviations of the data from the ideal P-T trace. Thus, the amount of variance accounted by the model is itself an indicator of anomalous

system behavior. The remaining figures show how (conceptually, at least) it is simpler to detect specific anomalous behavior from the residuals (model minus data).

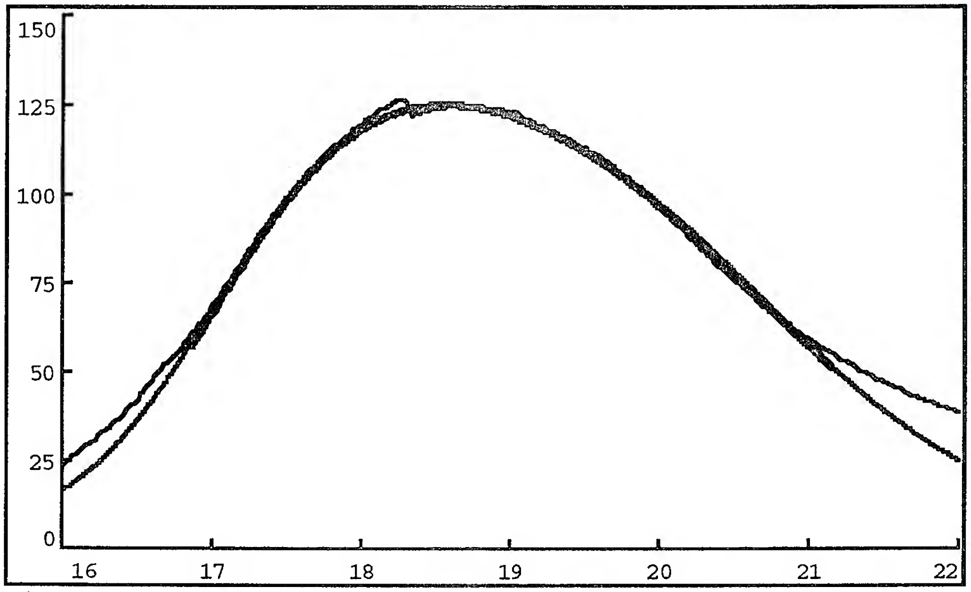


Figure 12 - Model fit to top-of-curve focus of data exhibiting blowby error. The heavy line represents the model fit on the focus interval. This interval is dynamically determined from the initial, full-interval model fit.

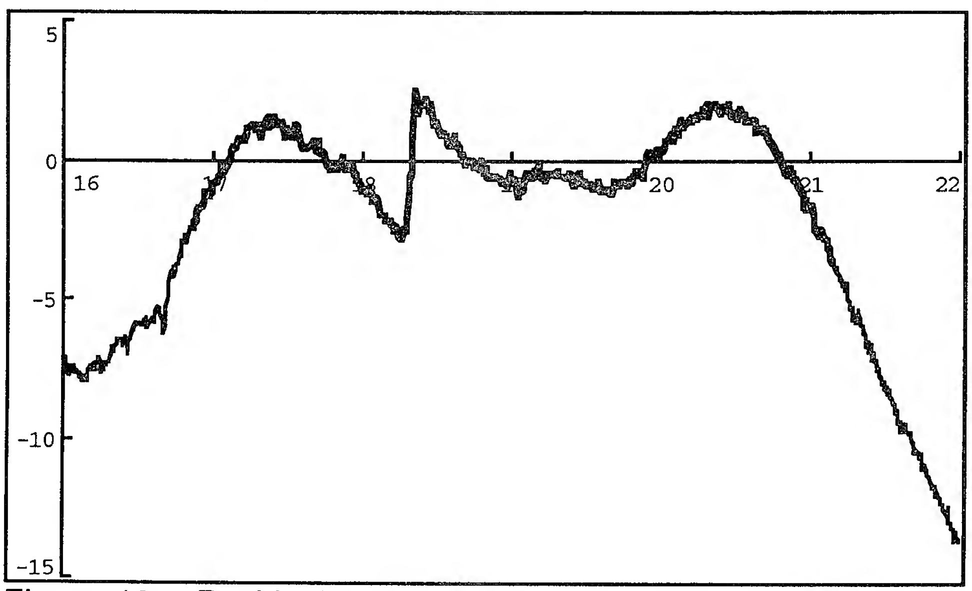


Figure 13 - Residuals plot derived from Figure 12. Note how blowby artifact is translated to an aberant, isolated crossing of the horizontal axis.

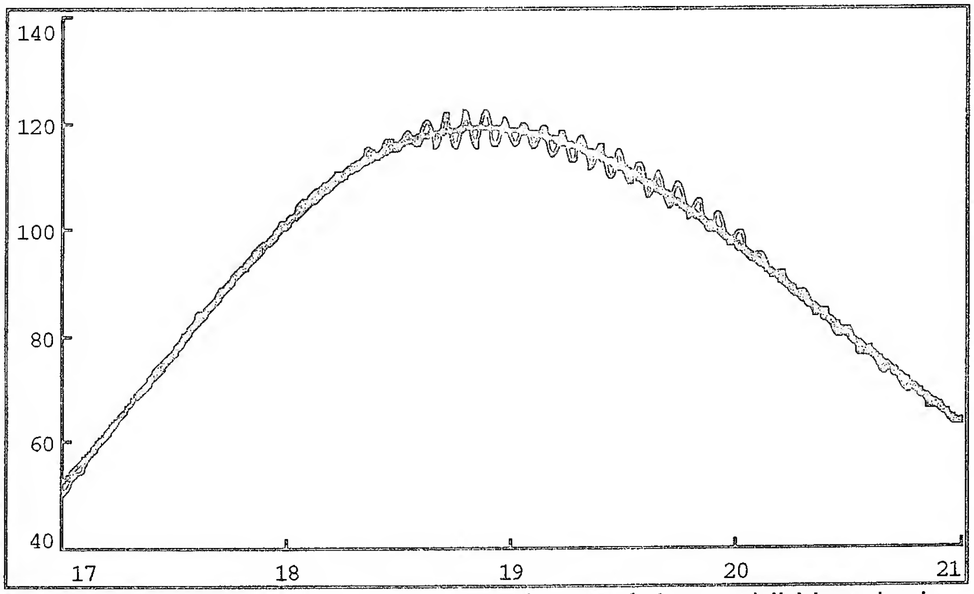


Figure 14 - Model fit to top-of-curve focus of data exhibiting ringing error. The heavy line represents the model fit on the focus interval. This interval is dynamically determined from the initial, full-interval model fit.

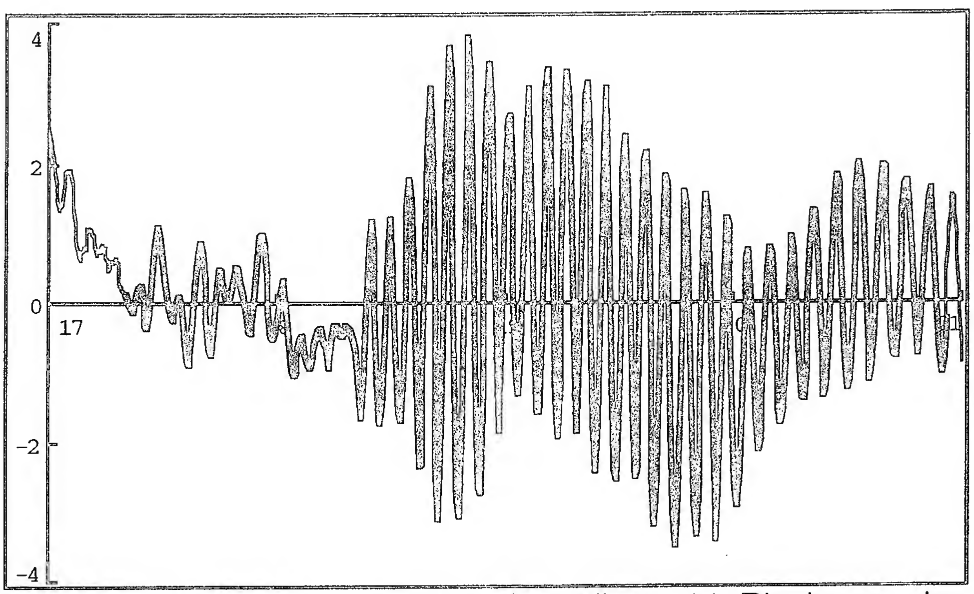


Figure 15 - Residuals plot derived from Figure 14. Ringing can be reliably detected simply by characterizing the axis crossings of the residuals plot.

5. CONCLUSIONS

In general, the simple seven-parameter, difference-of-sigmoids model performed better than anticipated in modeling the various data sets. In fact, it was anticipated (and hoped) that less variance would be accounted by model fits to data sets exhibiting particular fault types. Significant deviations from the expected variance accounted in normal data sets only occurred on narrow FOAs in which a connector fault generated a block-like data trace.

In particular, the model fit was exceptionally tight on the ascending limb. However, it was observed that the model fit was frequently poor on the lower portions of the descending limb - the P-T "tail". This was due to a lack of symmetry between the tail portion and that part of the descending limb immediately following the data peak. Later work has identified this region as coinciding with the exit of the shell from the weapon muzzle - this results in a drastic change in pressure-model dynamics.

Proof that significant automated diagnosis capabilities can be derived from a relatively simple system that matches a generic model to a P-T data set, and then applies specialized fault diagnoses procedures to the model-data residuals is evidenced. This procedure is modular: the model match can be revised without affecting the specialized fault detectors, and each fault detector can be modified without affecting the performance of the remaining AFDPS components.

This modular approach will support further development efforts. In particular, this approach directly supports a general graph-understanding methodology that can be applied to any problem in which interpretation of graphical data is an integral part of a decision analysis process.

Success for the graphical understanding approach will allow the automation of the transducer fault diagnosis process, and thus eliminate the requirement of having a data interpretation expert on-site or on-call during ballistic chamber testing. A fast, automated turn-around of analysis results will allow the transducer installation personnel to operate more effectively.

REFERENCES

Dayhoff, Judith E., Neural Network Architectures, An Introduction, Van Nostrand Reinhold, 1990.

Fleming, Gary C., Automatic Fault Diagnosis Prototype System (AFDPS), Final Report, Prepared for Instrument Development Division, Combat System Test Activity, Aberdeen Proving Ground, 1992.

Press, Flannery, Teukolsky, and Vetterling, Numerical Recipes, The Art of Scientific Computing, (Chapter 10.6 - Conjugate-Gradient Descent), Cambridge University Press, 1986.

Wasserman, Philip D., Neural Computing, Theory and Practice, Van Nostrand Reinhold, 1989.

(1)

Analysis of Simulations and Measurement Data Through Data Visualization

(2)

Jody Smith*
David Blair
Stephen Clanton
Loren Dickerson
Jill Gothart
Kenneth Green
Cliff Liles
Anna Norton
Mark Paris
Garth Powell
Randy Wood
Robert Oravits

(3)

SY Technology, Huntsville, Alabama 35816*
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SFAE–PEO–MD–TSD–TS, Huntsville, Alabama 35816

AdViSOR
(Advanced Visualization of Simulation Or Reality)

AdViSOR(Advanced Visualization of Simulation Or Reality) is a general purpose data visualization application which runs on a Silicon Graphics workstation. A companion code to the ThOR(Theater Optical and Radar) signature simulation, AdViSOR provides for the visualization of all data generated within ThOR with line plot displays, rendered object displays, and image displays. Input data may be entered directly from the ThOR output file, a previously created AdViSOR file, or from columnar ASCII files. Many standard image file formats may also be input as well as many unique formats used within the missile defense arena.

AdViSOR has been used in the analysis and study of test target signature phenomenology for PEO–MD, TTPO, Optical Discrimination Algorithm(ODA) Development Center, ARROW, and the ODES Demonstration

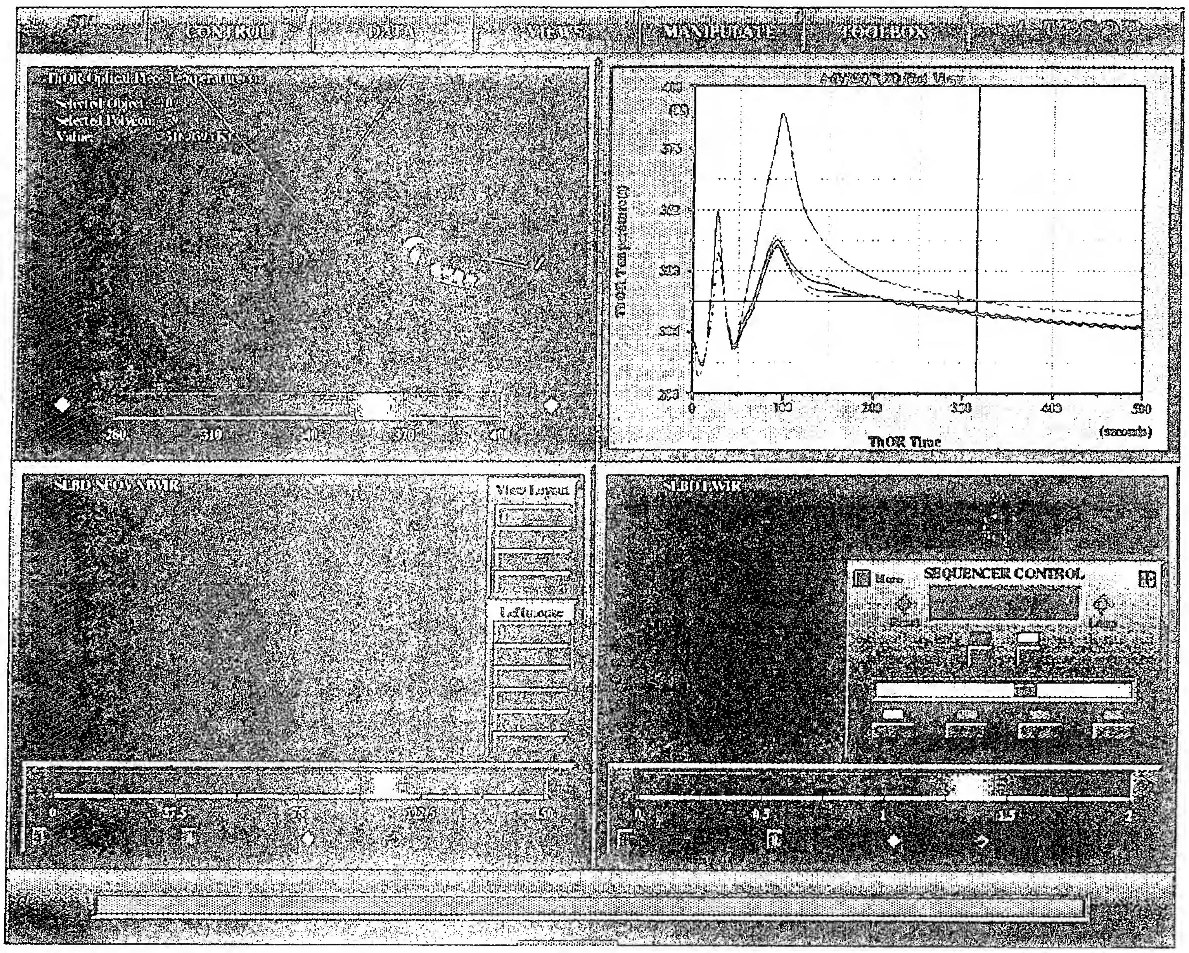


Figure 1. Typical AdViSOR screen with a rendered view, a 2D plot view, and two image views.

Flight(ODF) programs. Signature phenomenonology studies using AdViSOR have been conducted on infrared imagery/data from Sea Light Beam Director(SLBD), High ALtitude Observatory(HALO) InfraRed Imaging System(IRIS), Airborne Surveillance Testbed(AST), and Experimental Test Site(ETS) of Storm and Lance test targets.

As the name suggests, AdViSOR was written with the intent to display simulation data alongside test data in order to validate the simulation code as well as characterize the real system being simulated. Through different combinations of two–dimensional plot views, three–dimensional rendered views, and image views almost any data set may be visualized as shown in Figure 1. In addition, time sequential data sets plotted in various views may be animated synchronously with the aid of a sequence controller. One view is chosen to have the master time and all other selected views slave their time to the master. In this manner, time sequences of test data may be animated and compared to the corresponding simulation data.

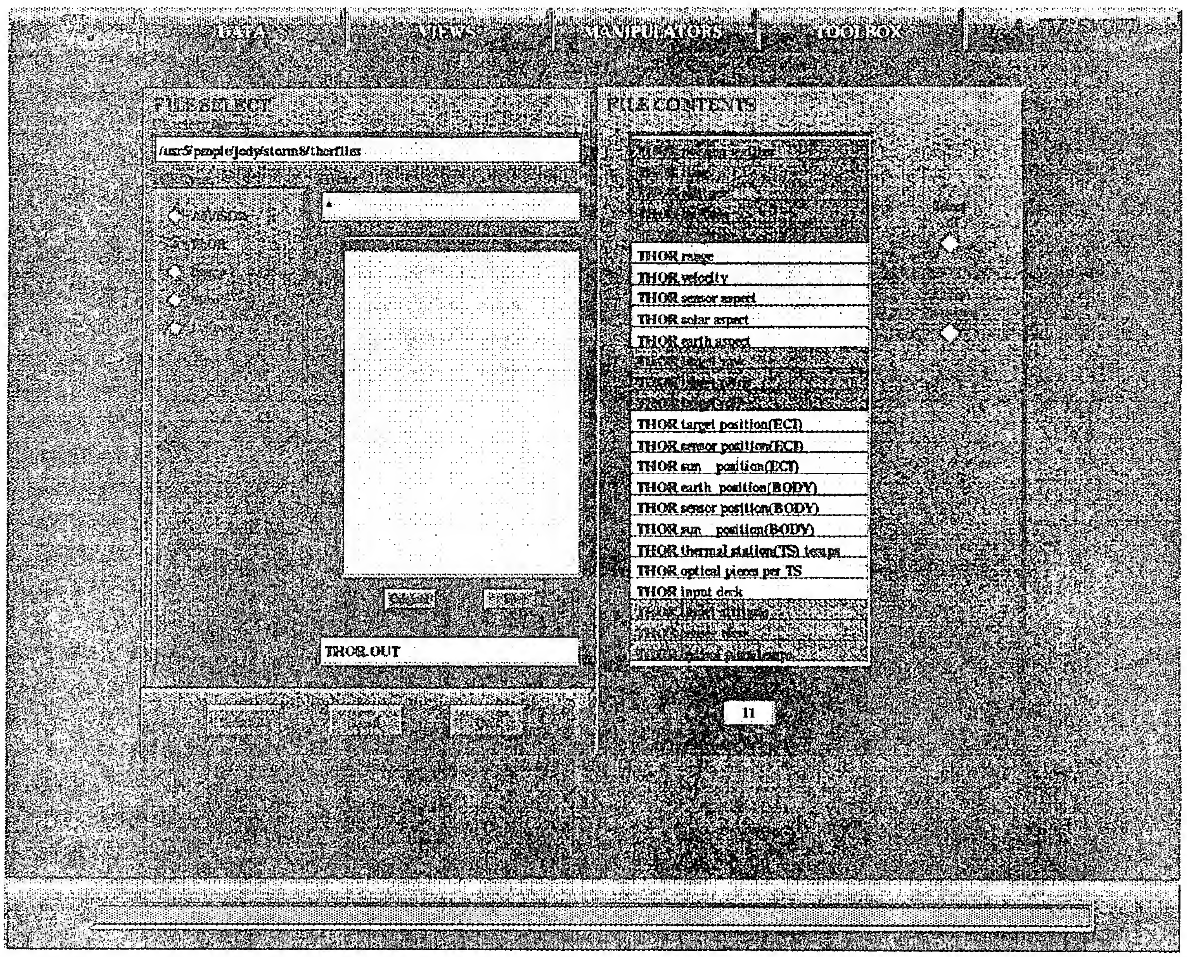


Figure 2. AdViSOR file input screen showing user selection of specific data items within file contents in order to maximize usage of computer memory and user's time.

In order to make the most efficient use of available computer resources, AdViSOR implements some unique features for the manipulation of data items. When loading data from an input file, the user may select only the data items within the file that are needed and load those into memory as illustrated in Figure 2. This feature also saves a considerable amount of time for the user when using large data files where only a smaller portion of the file is needed for analysis or visualization. All data items loaded into AdViSOR are organized into a standard format called a data module and placed within a clipboard list. A list of all data modules currently loaded into memory may be viewed in the clipboard viewer. The clipboard viewer lists the data modules with their data type, array dimensions, and source file. The user may rename, rearrange array dimensions, dump to an ASCII file, or delete data modules within the clipboard viewer. The clipboard implementation enables the user to display/visualize a data module concurrently in different views, possibly in different types of views (such as a rendered view and a 2D plot view) without duplicating data.

At present, three types of views in any combination may be created in AdViSOR (2D plot views, render views, and image views) with a maximum of sixteen views on the screen at one time. The 2D plot view provides a simple mechanism for visualizing two dimensional data where any dimension of any data module may be plotted against another. Within the 2D plot view the data may be autoscaled, scaled to discrete limits, or zoomed all under mouse control. Either line plots or point plots may be produced. The render view provides for the visualization of three dimensional geometry models with intensity or color scaled polygons. The attitude of the model along with a specific sensor or view perspective may also be initiated. Within the render view the model may be zoomed, rotated, or translated all under mouse control. The image view provides for the display of multiple images which may be autoscaled over each individual frame or over all frames within an array of images, or clipped to a user supplied intensity range. Several color look up tables are provided for the display of the images. The user with mouse control may arrange the images in any manner within the view changing their display size as well as cropping the image(s) to any size desired.

THOROUGHLY MODERN DATA ANALYSIS AND VISUALIZATION

Jon Wada Carolyn Boettcher

Hughes Aircraft Co.

ABSTRACT

This paper describes an ongoing effort to upgrade existing data analysis capabilities to provide a thoroughly modern data analysis and visualization facility that will support an automated, highly efficient process. Large volumes of instrumentation data collected in the radar integration laboratory or during flight test are typically stored on high speed tape recorders and analog videotape. Additional data is available in the form of notes and debrief summaries. This data must be accessed, analyzed and correlated to evaluate radar system performance, facilitate system verification and trouble shooting, and investigate the effect of new processing techniques and algorithms. Current data analysis software employs procedural code that is difficult to reuse, executes off-line, and cannot access multi-media data. In contrast, the system we are developing is envisioned to support a data analysis process as it "should be". That is, the tools eliminate or automate tedious, mechanical tasks; provide on-line visualization of all critical information; and are easily accessible to users and developers. The paper describes both our vision for the completed system and its current implementation and deployment on radar programs within Hughes.

INTRODUCTION

Performance evaluation is a critical step in the development of radar systems. For that purpose, Hughes has developed instrumentation and data analysis systems for all of its radar programs, including the F-15 APG-63 and APG-70, the F/A-18 APG-65 and APG-73, the F-14 APG-71, the B-2 ALQ-181, and various classified programs. Over the life of these programs, Hughes has developed tools to help in reducing and analyzing data. Typically, older tools are hosted on mini- computers, execute in batch mode, and produce

off-line graphics. The analysis software is often difficult to reuse and for these older tools, there was no defined reuse process.

The data analysis process supported by these older tools is extremely labor-intensive; very little is automated. Information about flight test configurations and content is not readily available to analysts. Documents are either paper or in electronic formats that are not easily shared. Analysis applications, most often implemented in FORTRAN with primitive tools, are built with no standards for architecture or interfaces and no planning for reuse. In addition, the platforms on which these tools are hosted have insufficient power for large scale simulation.

Several years ago, we began to upgrade the existing data analysis tools and processes by taking advantage of the latest COTS technology, including work-stations, interactive visualization, multi-media capabilities, and COTS software tool kits that facilitate reuse and provide visual programming. The goals for the new data analysis system include:

- **Eliminate tedious or mechanical tasks** by using tool kits, a standardized application programming interface (API), application frameworks, and graphical user interface.
- **Facilitate access to critical information** by developing database links to critical information, providing tools for searching and organizing data, and exploiting interactive visualization with color, three dimensional graphics and online video and animation.
- **Support large-scale simulation** and extensive post-processing analysis with toolkits and a standard simulation architecture.
- **Integrate many facets of radar development** including rapid prototyping and simulation, test planning, data analysis, and problem administration.
- **Provide open access to data and analysis software** for users and developers.

This paper reports on progress we have made in exploiting state-of-the-art COTS technology to automate much of the manual data reduction and analysis process, dramatically decrease data analysis cycle time, and increase analyst productivity.

BACKGROUND

The Advanced Data Reduction and Analysis System (ADRAS) project was begun in 1991 in response to the anticipated needs of several new programs. A review of the analysis process and tools initially proposed for these programs revealed that the quantity of data to be analyzed and the required analysis cycle times were more than could be accommodated by then-current tools and processes. Analysis of a single, two-minute radar test on one project, for example, might require as much as several months to complete.

In response, a survey was undertaken of COTS high performance graphics workstations and Silicon Graphics was selected as the initial platform to host ADRAS. An initial version of ADRAS was developed and released in July, 1992 for use on a new radar development program with a second version released in March, 1993. This second version is now being successfully used on that program. Enhancements to ADRAS and corrections of bugs has continued to the present.

The success of ADRAS prompted us to consider its widespread deployment across all Hughes radar programs. As a result, ADRAS is currently being used on two other radar programs and has been targeted and demonstrated for a third program.

ADRAS ARCHITECTURE

The current version of ADRAS is hosted on SGI Indigo and Indigo 2 workstations running at 100 to 200 megaHertz under IRIX 5.2, SGI's version of UNIX. Each workstation typically has 64 megabytes of RAM, 3 to 5 gigabytes of disk, and a 3-D accelerator graphics board. Each user has a 20 inch monitor with 1280x1024 pixels and 24 bit color. Workstations communicate with a SUN/490 or Silicon Graphics Power

Challenge XL server over Ethernet. Instrumentation data is made available to each workstation from the server. To store instrumentation data from multiple flight tests, a tape juke box sits on the server. A single juke box may store 6 terrabytes of data or more. Figure 1 illustrates a typical workstation configuration for ADRAS. The figure also shows the interface to the actual embedded system through a Universal Interface Adapter (UIA) that sends data to four DCRSi units capable of recording radar data at a combined 50 megabytes per second. The four data streams are merged by a Multi DCRSi Interface (MDI) and stored on the juke box or sent directly to the server.

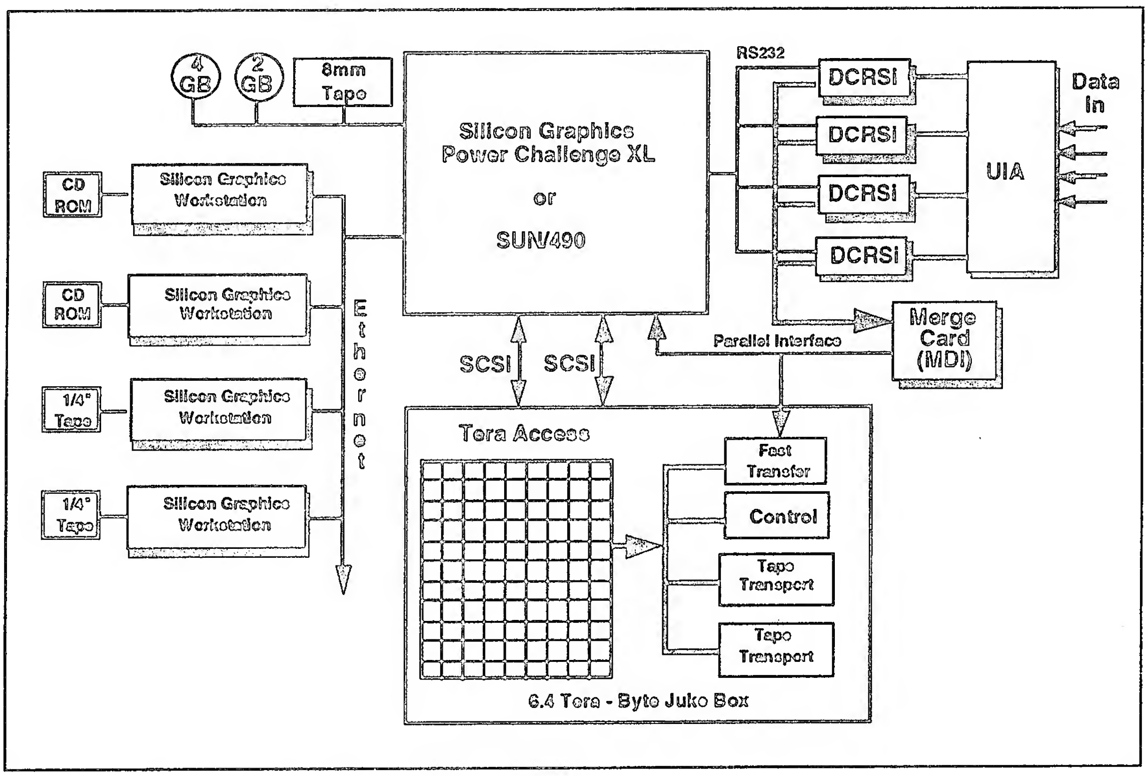


Figure 1. ADRAS Workstation Configuration. Analysts have immediate access to multi-media instrumentation data.

Consisting of approxmiately 28,000 lines of code, the ADRAS software is implemented using SGI's port of the AT&T C++ Language System, Release 3.0. ADRAS also makes maximum use of COTS software available on the SGI platform. As currently implemented, the ADRAS software architecture has three layers, illustrated in Figure 2.

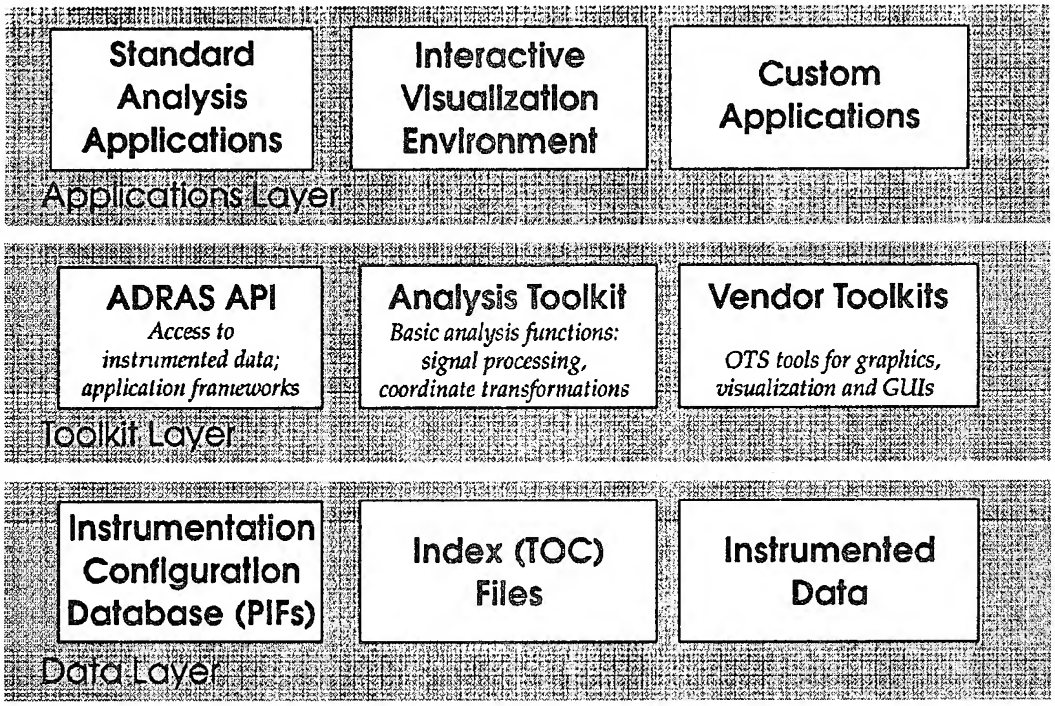


Figure 2. ADRAS Software Architecture. The three layers of the ADRAS architecture makes maximum use of COTS components.

1. **Data Layer** includes instrumented test data, index files for searching the data, and a set of relational data bases called Primary Instrumentation Files (PIFs) that map from the Operational Flight Program (OFP) variables into the instrumented data.
2. **Toolkit Layer** includes an analysis tool kit of reusable classes, the standard UNIX and C++ libraries, the ADRAS API, and API's to selected COTS tools.
3. **Application Layer** includes a standard set of analysis applications, interactive visualization modules for use with SGI's IRIS Explorer environment, and custom analysis applications built by ADRAS users to perform project and radar mode specific tasks.

The data layer is composed entirely of custom software whose purpose is to organize and provide access to instrumentation data. Figure 3 illustrates the schema or relationships between the various types of data files.

- **Raw Instrumentation Data** is partitioned into standard UNIX files of approximately 100 megabytes length.

- Index Files provide access to the raw instrumentation data. Automatically generated table of content (TOC) files enable efficient searching on time code, bus source, and block identification. Catalog files can be created by users to create custom sets of data and TOC files.
- PIF (Primary Instrumentation File) Database provides the information to parse instrumented variables. It is generated automatically from the OFP.

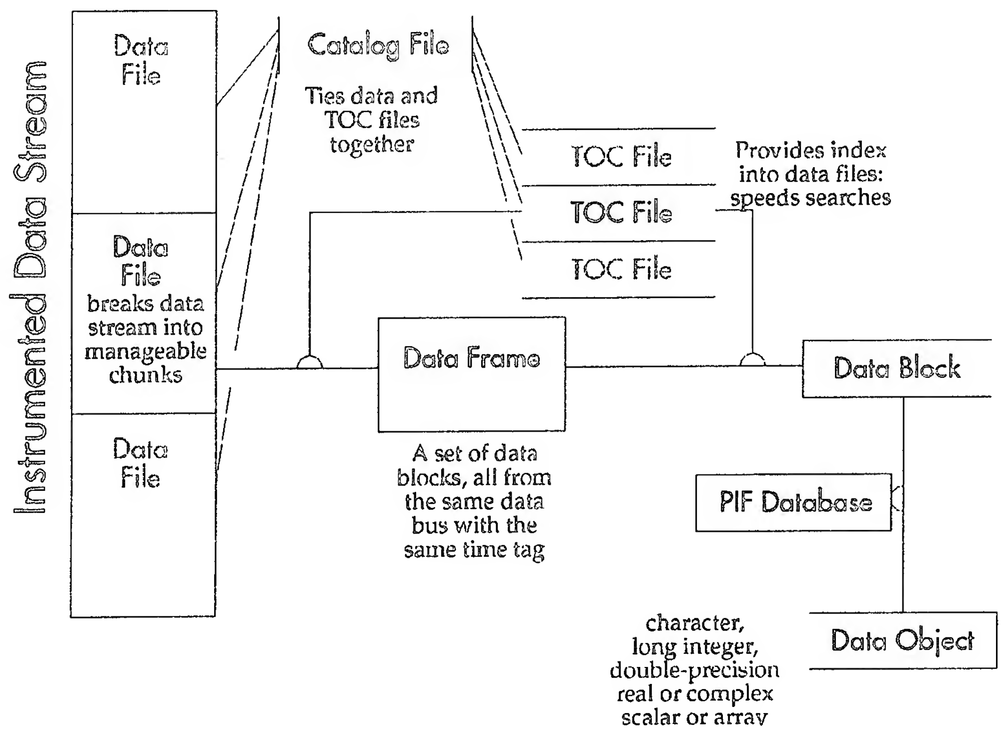


Figure 3. ADRAS Data Schema. The schema for the data layer enables symbolic access and fast searching of instrumentation data.

The toolkit layer is composed of both COTS and custom software.

- ADRAS API is a C++ class library that defines the ADRAS standard classes and abstract data types that are linked into applications to enable access to instrumentation data.

- **Analysis Toolkit** is a C++ class library that includes useful classes such as time of day, linked lists, coordinate transformations, etc.
- **COTS Toolkit** includes standard UNIX and C++ libraries, Motif, and SGI's Inventor and Explorer for interactive 3-D graphics and scientific visualization.

ADRAS VISION

ADRAS employs several implementation strategies to achieve its vision of a thoroughly modern data analysis. Software portability and extensibility are achieved by using a layered architecture, modularity, and object orientation; by planning for multiple target platforms such as SGI, Sun, HP-UX, MacOS and Windows/NT; by emphasizing commercial standards; and by using off-the-shelf components wherever practical. The approach is based on an open system architecture that employs standard protocols to enable inter-application communication and messaging across networks, even for multi-vendor systems; and a layered architecture that will allow system components such as operating systems and user interfaces to be changed without affecting the entire system.

Some of the standards used in implementing the current version of ADRAS or planned for future versions include:

- **X/Motif** for windowing, graphics and user interface
- **SQL** for database queries and transactions
- **TCP/IP and NFS** for network communication

Key technologies that are essential to the success of ADRAS include: information systems such as database management, document and image management, online video and communications; data presentation and user interfaces, including graphics, scientific visualizations, and graphical user interfaces; and radar modeling and application development, including rapid application development, a standard simulation architecture, application frameworks, modeling and large-scale simulation support. All of these

technologies are built on top of a computing infrastructure that includes mass storage, networking, and inter-application messaging.

Using these technologies, we believe that a typical analysis application can be built quickly from a library of layered, reusable components. The layered architecture enables problem-domain experts to concentrate on their core expertise and relieves them of concern with detailed data formats and data extraction methods. Moreover, developing a simulation framework and reusable libraries of common radar algorithms will further relieve the analyst from repetitive effort in developing new mode simulations.

VISUAL PROGRAMMING

COTS interactive visualization tools such as Mathematica, MATLAB, and SGI's IRIS Explorer can also speed analysis and algorithm development. As an example, the Explorer is a visual programming framework for assembling modules into application maps that is being used at Hughes on several radar programs. In Explorer terminology, a *module* performs a single task such as synthesizing or retrieving data, computing FFT's or noise estimates, graphing or rendering. A *map* is an application that performs an analytic task such as simulation, statistical analysis, etc.

In the Explorer environment, control panels and editable inter-module connections enable analysts to alter parameters and configurations on the fly. The visual, interactive Map paradigm with its drag-and-drop connection makes it easy to reconfigure and adjust processing flows. The control panels allow easy adjustment of processing parameters. An Explorer Map looks like a block diagram, the way many engineers like to think about systems. Documentation is accomplished effortlessly via snap shots of screens. Finally, interactive, 3-D graphics is available for effective presentation of information. Interactive manipulation of 3-D images, including real-time rotations and translations greatly enhances the usefulness of data.

Explorer integrates easily with other ADRAS analysis tools and modules. The highly interactive, visual Explorer programming paradigm facilitates marked improvements in analysis cycle time. Explorer's modular approach allows easy reconfiguration of algorithms. It lets the analyst experiment with algorithmic variations with a minimal amount of programming effort.

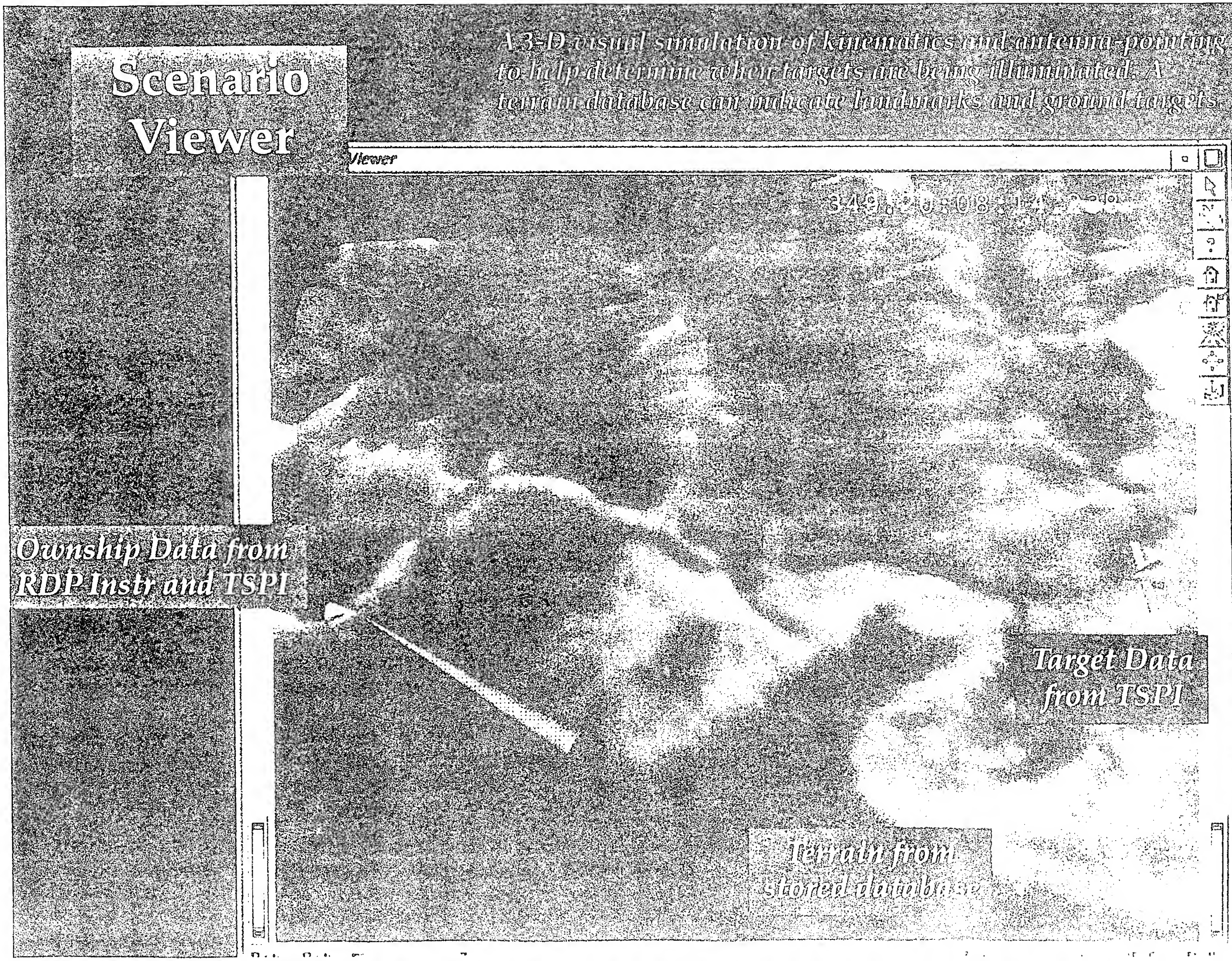
ONLINE VIDEO AND ANIMATION

ADRAS includes support for online access to cockpit video that is recorded during flight. Online video access can increase productivity by eliminating contention for access to videos and VCRs, by reducing the time spent searching for videos, by allowing analysts to review videos at their workstation where they have immediate access to other analysis tools, and by facilitating frame-by-frame viewing of videos and capture of still images for study and presentation.

ADRAS also includes support for 3-D animated simulation of ownship kinematics and antenna pointing. It is envisaged that ADRAS will have access to terrain databases that can be overlaid with target and ownship data. For example, figure 4 illustrates a scenario where the antenna is being pointed to illuminate a ground target. As shown, ownship's antenna beam on the left of the map is not illuminating the target to the far right of the map. The simulation can help determine when the target is in the antenna beam.

LESSONS LEARNED

For many of the ADRAS developers and users, this was their first experience with UNIX, C++, and the object-oriented programming paradigm. Analysts who desired it were given 20 hours of class room instruction in C++ held after working hours. Otherwise, analysts were given a copy of Stan Lippman's *C++ Primer*. Use of class libraries and abstract data types was taught by providing sample code and view graph presentations, and by informal one-on-one instruction. Training in Unix was provided by


Scenario Viewer
A 3-D visual simulation of kinematics and antenna-pointing to help determine when targets are being illuminated. A terrain database can indicate landmarks and ground targets.
Viewer
Ownship Data from RDP Instr and TSPI
Target Data from TSPI
Terrain from stored database

making available a copy of Kaare Christian's *The Unix Operating System* and Kernighan and Pike's *The Unix Programming Environment* and by much one-on-one instruction.

Analysts' assessment of the adequacy of this training was mixed. Some analysts felt the need for some formal training in the Unix environment. In addition, since this was the first C++ experience for most of the analysts, many of them do not fully understand the software architecture. This makes it more difficult for them to fully utilize the libraries when developing applications and to debug their applications. However, the ready availability of programming assistance mitigated this difficulty to a considerable extent.

In the future, a combination of training and programming support will be necessary to ensure that analysts can make full use of data analysis tools which are growing in complexity and sophistication. Training must be matched with the analyst's interest and ability. Programming support must be expert and immediately available. Delays compound analyst frustration and degrade problem response time.

The ADRAS API was still being developed as analysts began to use the system. Although this caused some frustration, we believe that this real-world beta testing resulted in a very usable set of tools. As a result of this experience, we believe that end-user feedback into the design of user interfaces is essential and must be based on real-world usage.

A final lesson learned is the cost effectiveness of investing in code development tools. They will pay for themselves in programmer productivity and code quality. For release 2.0 of the ADRAS API, almost 20,000 lines of code were written and tested with only 29 bugs discovered by users. This was largely due to the exceptional suite of development tools that were available.

FUTURE WORK

When completed, ADRAS will include several information based tools. These include flight-test database query and data entry applications; document managers and viewers,

including on-line process documentation using HTML; and communcation tools such as multi-media E-mail, whiteboarding, and Usenet-style news groups. More extensive signal- and data-processing functionality will be added to the IRIS Explorer module set, and that module set may be ported to similar environments such as AVS/Express and Khoros as well. Integration of analysis applications will allow users to query a database of flight-test descriptions in order to select a test event of interest, view the cockpit video for that test in order to isolate a specific radar event (e.g., a false detection or apparent system failure), and then retrieve and analyze the appropriate radar instrumentation data, all within an integrated graphical user environment.

CONCLUSION

The ongoing ADRAS project is bringing higher levels of automation to the process of analyzing the large volumes of information typically collected during radar system integration and flight test. When ADRAS is fully implemented and deployed and analysts have been fully trained in using its features, we expect the turnaround time for identifying and fixing radar performance problems to be significantly reduced. ADRAS has already achieved significant advances over the older data analysis systems by putting all relevant information online at an analyst's workstation and providing a visual programming environment for reuse and rapid prototyping of algorithms.

WEAPON SOFTWARE MANAGEMENT SUB-WORKING GROUP (WSMSWG)

Co-Chairs:
Don Rauch and Paul Storey

THE ECONOMIC BENEFITS OF SOFTWARE PROCESS IMPROVEMENT

Kelley L. Butler

OC-ALC/LAS
3660 C Avenue, 2S12
Tinker AFB, OK 74145-9144

Overview

This paper discusses a study that was performed by Software Productivity Research at the Oklahoma City Air Logistics Center Directorate of Aircraft Software Division's (OC-ALC/LAS) Test Software Branches from April to June 1994. The goal of the study, performed under contract to Mr Lloyd Mosemann, the Deputy Assistant Secretary of the Air Force for Communications, Computers, and Support Systems (SAF/AQK), was to determine the economic benefits of software process improvement. This paper will not attempt to recreate Software Productivity Research's report, instead it will provide an overview from the perspective of the organization that was studied.

Introduction to the Organization

The Oklahoma City Air Logistics Center (OC-ALC) is located at Tinker Air Force Base. The Software Division, composed of over 400 personnel in seven branches, is responsible for the development and maintenance of many different Air Force software items.

The Test Software Branches develop and maintain Test Program Sets (TPSs) which are used with Automatic Test Equipment (ATE) to provide repair capability for complex avionics. The Industrial Plant Equipment Branch, provides software support to jet engine, constant speed drive, and eddy current testing along with the support of the software for several automated processes associated with jet engine overhaul. The weapon systems affected by the software products produced and maintained by the Test Software and Industrial Plant Equipment Branches include the A-10, B-1B, B-52, C-141, C-5, E-3, (K)C -135, F-15, F-16 aircraft and six engines.

Three branches develop and maintain Operational Flight Program (OFP) software for the E-3, B-1B, B-52, Air Launched Cruise Missile (ALCM), and Advanced Cruise Missile (ACM).

The Management and Technical Support Branch provides the personnel, training, funding, drafting, computer, and other support functions.

The study focused on four Test Program Set projects located in the Test Software

Branches. The first project began in March 1986 and was completed in May 1989 and the last project started in June 1992 and was completed in March 1995.

Hereafter, the test software branches will be referred by the Air Force organizational mnemonic, LAS.

Software Productivity Research

Software Productivity Research (SPR), the contractor hired by SAF/AQK to perform the study, is a leader in the software management and measurement industry. One of SPR's products, Checkpoint, a knowledge based tool that estimates, measures, and assesses software environments, was used for the study. Checkpoint's knowledge base is composed of data from over 4,700 software projects. The knowledge base can be used to assess a project against industry or internal standards for cost, quality, schedule, and productivity. The Checkpoint knowledge base allowed SPR to make comparisons among the four LAS projects used in the study.

Software Process Assessment History for the Test Software Branches

LAS performed their first software process assessment in 1989 when they were rated a Software Engineering Institute (SEI) Maturity Level 1. In 1990, LAS was chosen by the SEI to be an alpha site for the updated software process improvement methodology. That assessment, performed during March 1993 by personnel from the SEI, resulted in LAS being rated as an SEI Capability Maturity Model (CMM) Level 2, the first in the Air Force.

LAS is currently working on achieving a SEI CMM Level 3 with the next assessment scheduled for 1996. SAF/AQK has set a goal of an SEI CMM Maturity Level 3 for all Air Force Software organizations by 1998.

Process Improvement Approach

LAS has been working on process improvement since 1986 and began their relationship with the SEI in 1989. The SEI Capability Maturity Model (CMM) has been the basis for the process improvement efforts since its release in 1991. To facilitate process improvement, LAS has developed and implemented a process improvement infrastructure whose purpose is to guide and monitor the organization's process improvement efforts.

The LAS process improvement infrastructure includes the Management Steering Team (MST), composed of the senior organizational management along with the Software Engineering Process Group (SEPG) composed of technical personnel. Additionally, Technical Working Groups are established, as required, to work specific areas.

The MST and the SEPG work together to insure that the improvement efforts are effective and that they are solving the problems facing the organization. The SEI CMM is used to guide the process improvement efforts. Yet, while many of the improvements that are worked by LAS can be directly traced to the CMM, there are also many that are worked because, while they may not be directly traceable to the CMM, they are issues to the organization and must be resolved. This approach, work all issues, not just CMM issues, is one of the reasons that LAS feels they have had success in their process improvement efforts and why, as this report will detail, they have seen an impressive return on their process improvement investment.

The biggest key to LAS's success is the Management Steering Team and the focus and attention that they bring to the process improvement efforts. Nothing can replace top management leadership and support. The MST meets monthly, with the meetings often lasting three to four hours. Process improvement status is briefed, issues are resolved, and resources are assigned to the improvement efforts. The actions of the MST send a clear message to the LAS employees that process improvement is important.

Another key to success is the funding for process improvement that has been provided by Air Force Material Command. The funding is significant; it allows the improvement efforts to be worked at the same management attention level as the organization's other workloads. The funding that the Air Force is providing for process improvement is one of the reasons that a study of this type was needed. Many organizations are making large investments in process improvement and it is important that the benefits of those investments are quantified. Organizations may not show a quantifiable return during the early process improvement efforts, but as they progress up the SEI CMM Maturity scale they should be able to show the benefits of their efforts. There are several different methods used to show the benefits of process improvement, not just the one that is outlined in this paper; others are outlined by the SEI in "Benefits of CMM-Based Software Process Improvement: Initial Results," Technical Report CMU/SEI-94-TR-13. The bottom line is that while there may not be a standard definition for Return on Investment (ROI), every organization that is investing money and personnel in process improvement should be asking themselves, "What am I seeing in return?" They need to look for both the quantifiable and unquantifiable benefits.

Study Objectives

SAF/AQK approached LAS in 1993 to ask if they would be willing to be the subject of a study to determine the economic benefits of software process improvement. The goal of the study was to make a business case for process improvement. There is a tremendous focus on SEI Maturity Level but, in today's increasingly competitive world, many are asking what the real benefits are, what is the bottom line, how will my organization benefit from process improvement?

This study, to the best of LAS's knowledge, was the first independent study of any organization's software process improvement Return on Investment (ROI). *Independence* was key. Many groups, including LAS, have been reporting ROI data for several years yet

some critics were able to discount the reports as being self-serving for the organizations reporting the savings. SAF/AQK wanted a view, independent of the Air Force and the SEI, that showed the impact of CMM-based process improvement.

Project Selection

Four LAS projects were selected for the study. Each involved the development of Test Program Set (TPS) software to test avionic circuit boards for three aircraft and one jet engine. The projects spanned an eight year period from 1986 to the present. It was important that the projects selected were enough alike to invite comparison and that they cover a wide range of time to show the impact of the organization's process improvement efforts.

Project 1 (Baseline Project): C-141 All Weather Landing System (AWLS), 1986-1989

Project 2: B-1B Electrical Multiplexing System (EMUX), 1986-1988

Project 3: C-5B Automatic Flight Control System (AFCS), 1987-1990

Project 4: F110 Digital Engine Controller (F110-GE-129 DEC), 1992-1994

Study Methodology

As was stated above, this paper will not go into detail about the study methodology and data collected. That is contained in the report prepared by SPR for SAF/AQK, but a brief outline of the methodology used and data collected follows.

The study involved both interviews and data collection. The contractor conducted interviews with the project managers, project personnel, and LAS customers. Project personnel were asked to complete, as a group, two questionnaires. One was the Software Productivity Research Checkpoint questionnaire which looks at four major areas: personnel, process, technology, and environment. The second questionnaire, which SPR constructed especially for the study, concerned the SEI CMM Level 2 and Level 3 Key Process Areas. Each project team was asked to answer the questions in reference to how well the Key Process Areas were performed for that project.

In addition to the two questionnaires and the management and customer interviews, SPR also collected data on each of the four subject projects. The data collected included: project complexity, cost, schedule, and size (in source lines of code).

The contractor used the data collected in the interviews and from the projects along with the SPR Checkpoint tool to determine the impact of LAS's software process improvement efforts.

Study Results

Table 1 outlines the results of the SPR study. SPR noted improvements in four areas: process improvement ROI, defect rates, maintenance costs, and productivity.

The analysis performed by SPR indicates that the investment of $1.5 million dollars over an eight year period resulted in a cost savings of $11.3 million dollars when the three subsequent projects were compared to the baseline project for a ROI of 7.5 to 1. SPR arrived at the $11.3 million dollar figure by determining the additional amount the three subsequent projects would have cost had there been no improvements in productivity.

The study also showed that defect rates from the baseline project to the second project, the only other project for which LAS currently has ample defect data, had been reduced by a factor of ten. The baseline project experienced 3.39 defects/thousand source lines of code (KSLOC) compared to 0.28 defects/KSLOC for the second project, the B-1B TPS project.

Additionally, data provided by one of LAS's customers showed that in the past two years, LAS has reduced the cost of a TPS maintenance correction by 26%. Although maintenance was not a focus of the study, it is a large part of LAS's workload and therefore this data was very encouraging. Another important aspect of the improvement in the TPS maintenance correction process is that some have argued that the SEI CMM cannot be effectively applied to maintenance organizations. LAS has never believed that argument, in fact, it is stressed to LAS employees that maintenance actions must be treated as "mini" developments. All phases of the process have to followed if a quality product is going to be produced.

The most recent TPS development project is ten times more productive than the baseline project. This is attributed to improvements in both process and technology and while the effects of each cannot be separated, both SPR and the customer felt that the project benefited greatly from the LAS process improvement efforts.

Using the data from the two questionnaires completed by the representatives from the four projects, SPR developed rankings for the SPR and SEI Levels for each of the four projects. Each group of project representatives were asked to answer the questions from the perspective of how the activities were performed for that project. The SPR questionnaire looks at areas such as process, personnel, technology, and environment, while the questionnaire used for the SEI Level looked at the SEI CMM Level 2 and 3 Key Process Areas, including project planning, quality, and training. Analyzing the questionnaire data, SPR was able to show that, starting with the baseline project, each successive project had improved its SEI and SPR rankings indicating that LAS had successfully implemented a continuous process improvement program.

Category	Result
Return on Investment	7.5 to 1, An investment of $1.5 million dollars resulted in a savings of $11.3 million dollars
Defect Rates	10X reduction from the baseline project to the second project
Maintenance Costs	26% reduction in the average cost of a TPS maintenance action over last two years
Productivity	10X increase from the baseline project to the most recent project

Table 1: Benefits of Software Process Improvement.

Should Other Organizations Expect These Results ?

Would every organization that has implemented process improvement and moved up the SEI CMM Maturity Model show the same results as LAS? Would their results be better or worse?

These questions are difficult to answer. Every organization is different; yet, when the SEI studied the benefits of CMM based software process improvement, using ROI information that had been reported by several organizations, they noted that the ROI ranged from 4 to 1 to 8.8 to 1, with the median being 5 to 1. The ROI of 7.5 to 1 that was independently determined for LAS by SPR is within this range.

Conclusions

The study was useful in many ways. For LAS, it showed that the process improvement efforts were having a positive impact on the organization. For other organizations, it is a data point that can be used to generate support for process improvement.

Many felt that LAS was taking a risk in allowing a contractor to independently verify the process improvement results. That may have been true, but it was a risk that had to be taken. The Air Force is making a tremendous investment in process improvement and, as with any investment, the returns have to be verified. LAS knew, intuitively, that their efforts had benefited the organization and they had been reporting basic ROI on their individual improvement efforts for several years, yet they welcomed an independent study to validate their efforts. Additionally, the study helped identify problems with LAS's data archiving methods and introduced LAS to new methods that can be used to monitor their process improvement ROI.

Another key element of the study was the involvement of the LAS customers. The customers appreciated the fact that their input was used in the study and SPR was impressed that the customers were able to provide data showing improvement in the LAS processes.

One key to LAS's success is that they have worked very hard to implement a process improvement infrastructure that responds to the true needs of the organization. The SEI CMM is used only as a guide. An improvement is only implemented if the organization sees that it will add value to the processes. Additionally, many improvements which cannot be directly traced to the CMM were very important to the organization and had major impacts on the improvement efforts. This is not meant to discount the CMM or the impact that the SEI has had on process improvement. LAS is very much an advocate of the SEI and the CMM. The point is that the CMM must be applied intelligently.

Process improvement must be an ongoing, continuous process. An organization will either continue to get better or they will backslide, it is not possible to stay the same. As more organizations increase their process maturity, more data will become available to show the gains that can be seen in quality and productivity. The SEI CMM Level is an important indicator, but it is hoped that more and more organizations will share their specific data with the rest of the world as several did in the SEI "Benefits of CMM-Based Software Process Improvement: Initial Results" technical report.

At the time this study was done LAS had been at SEI CMM Level 2 for about a year and the contractor, SPR, felt that much of the organization was operating at the SEI CMM Level 3. While this study was not an assessment of the organization in the SEI CMM sense, it did provide the organization with useful insight into the improvement efforts. The organization will continue their efforts to achieve SEI CMM Level 3 and the higher levels and they will continue to monitor the returns on their process improvement investment.

References

Herbsleb, J, et al, "Benefits of CMM-Based Software Process Improvement: Initial Results" Technical Report CMU/SEI-94-TR-13, Software Engineering Institute, August 1994.

Nowell, Jeff, "An Analysis of Software Process Improvement", September 1994.

Paulk, M.,Curtis, B.,Chrissis, M.B., et al, "Capability Maturity Model for Software, Version 1.1," Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, February 1993.

Paulk, M.,Chrissis, M.B., et al, "Key Practices of the Capability Maturity Model for Software,Version 1.1" Technical Report CMU/SEI-93-TR-25, Software Engineering Institute, February 1993.

Software Productivity Research Informational Material, no date.

Real-Time Support of the Verification & Validation of Multiple Radars in a Shared Facility

Allison Heaton

WR-ALC/LUE
226 Cochran St.
Robins AFB, GA 31098

Robert J. Endacott
Gary K. Miyahara*
Steve Tomashefsky

Hughes Aircraft Co.
P.O. Box 92426
Los Angeles, CA. 90009
M.S. RE/R8/N537*
gmiyahara@msmail4.hac.com*

Abstract

The APG-70 Radar in the F-15 Eagle and the APQ-180 Radar in the AC-130U Gunship have a high degree of LRU commonality despite the different missions of their respective weapons systems. The time critical nature of these radars and their diverse functionality have been addressed in an AF sponsored study[1] with Hughes Aircraft Company that has developed an approach to leveraging a common OFP support facility for both radars. Key issues include: differences in the missions of the weapons systems, the aircraft performance, the radar modes, and the avionics suite; and the nature of the radar environment and real-time radar return data generation. This paper outlines the issues encountered in the study's investigation, and the features of both the radars and the support facility that have influenced the design of a shared OFP test environment.

Introduction

Cost-effective weapons systems support is critical in the current fiscal environment, and has been a key concern in the deployment of the AC-130U Gunship. Since the Gunship's APQ-180 radar uses the same processor suite and OFP architecture as the F-15 Eagle's APG-

[1]This work is the subject of an on-going study sponsored by Warner Robins Air Logistics Center SOF Engineering through the Avionics Software Technology Support (ASTS) Program.

70 radar, an attractive opportunity exists to reduce cost by leveraging support systems.

The APG-70 Software Development Facility (SDF) is currently under development for the F-15 Avionics Integration Support Facility (AISF) at Warner Robins Air Logistics Center. The SDF provides a comprehensive set of OFP development, radar system test, and instrumentation data reduction and analysis tools in an integrated environment. The extension of this APG-70 SDF to accommodate the Gunship APQ-180 support requirements can potentially provide the Air Force with an exceptional, cost-effective multi-radar support capability.

System Differences

Table 1 compares the diverse missions and functions of the AC-130U Gunship to those of the F-15 Eagle. These diverse weapons system objectives drive differences in sensor systems as outlined in Table 2.

Table 1. The Weapon Systems Support Diverse Objectives

	F-15 Eagle		AC-130U Gunship	
Primary Mission	All weather, air superiority/defense fighter, long range air strike		Special Operations Forces special tactical missions, close range interdiction	
Relative Performance	Short mission, high speed, low altitude		Long mission, slow speed, medium altitude	
Weapons Deployment	Air-to-Air	AMRAAM AIM-7M, AIM-9 20mm Gun	Air-to Gnd	105mm Howitzer 40mm Cannon 25mm Cannon
	Air-to-Gnd	Various Bombs 20mm Gun		

Though wide variances in derived radar mission requirements are noted in these tables, the underlying hardware LRU's and software architecture are largely common between these systems. Figure 1 illustrates the basic hardware LRU's of the APQ-180 radar system, identifying LRU's that are uniquely affected by the Gunship upgrades. Many of the differences in sensor performance are achieved by the OFP in the radar signal processor (multiple processing elements controlled by a Mil Std 1750A array controller) and radar data processor (Mil Std 1750A). Table 3 contrasts the different radar modes supported by the radars' OFP. Although the

Table 2. Gunship Unique Radar Requirements

Tactics	Physics	Radar Sensor
Sidelook	Large clutter extent Large azimuth coverage	GMTI endoclutter processing, dual channel A/D, new outer az gimbal
Steep Lookdown	Small radar swath	1-10 El bars, 40 degree polarization
Long Mission	More velocity drift/ position error	GPS position accuracy for long map navigation
Slow Speed	Smaller Doppler shift	Longer map formation, more map overlay

two radars have many similar modes, the performance and implementation of these modes are significantly different.

The interfaces of the radar to the rest of the avionics in the Gunship are also significantly different from the F-15 and affect both hardware (e.g., a single mission bus with Mission Computer, and Inertial Reference System, message data vs. separate serial message buses in the F-15) and software (e.g., message format and content).

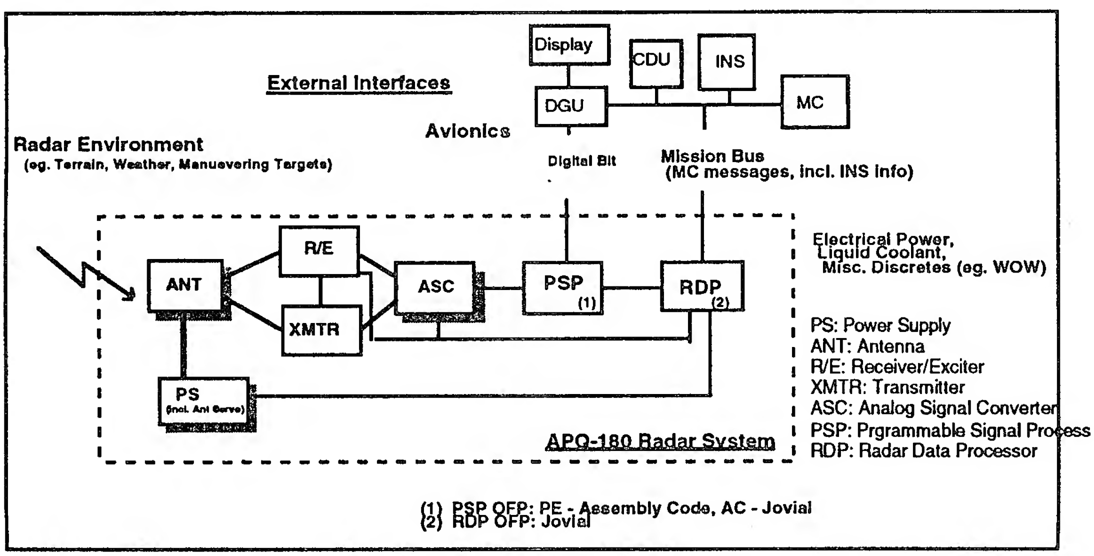


Figure 1. APQ-180 Radar System LRU's (shaded boxes: Gunship Unique)

Table 3. APQ-180 Tactical Modes

High Resolution Maps	Modified from APG-70
Air-to-Ground Range	Modified from APG-70
Real Beam Ground Map	Modified from APG-70
Weather Map	New for Gunship
Beacon Search (air-to-air and air-to-ground)	Modified from APG-70
Beacon Track (air-to-ground)	New for Gunship
Fixed Target Track	New for Gunship
Projectile Impact Point Prediction	New for Gunship
Ground Moving Target Indication	New for Gunship, (Now on APG-70)
Ground Moving Target Track	New for Gunship

Support System Impacts

Because of the high degree of commonality between the radars, the APG-70 SDF was used as the baseline for assessing the impact of radar system differences on support system requirements. The WR-ALC SDF is composed of four configuration items: the Central Development Facility (CDF), the APG-70 Radar Test Bench System (70RTBS), the Advanced Software Bench (ASB), and the Instrumentation Data Reduction and Analysis System (IDRAS). Figure 2 illustrates the support systems and their associated

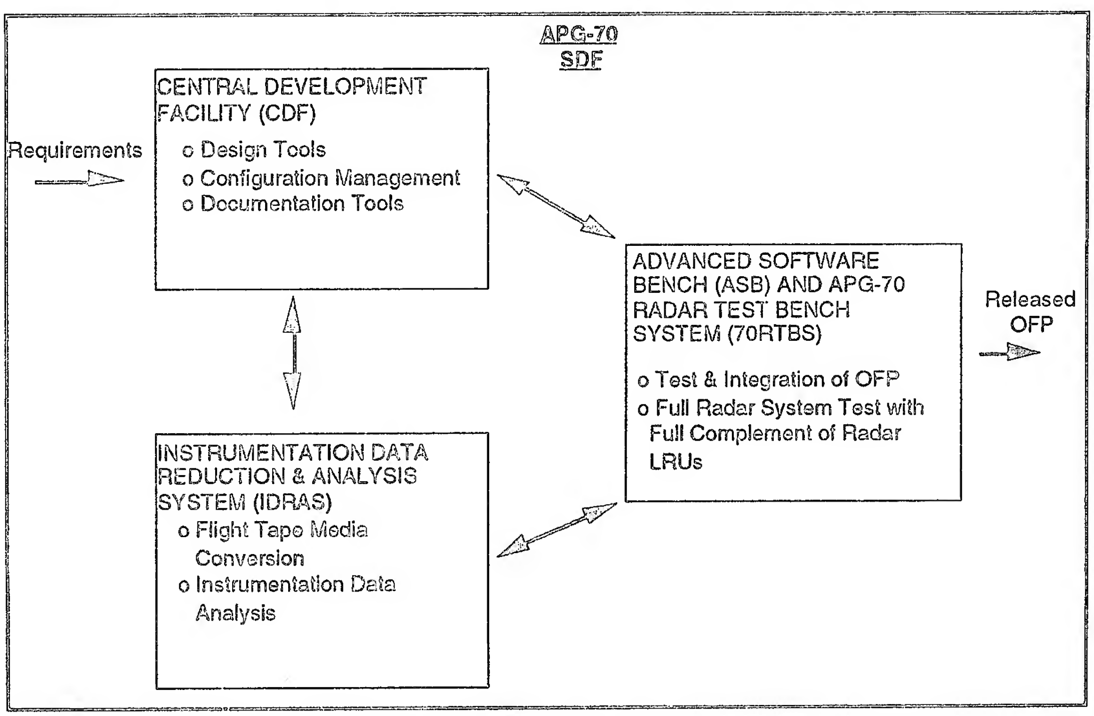


Figure 2. WR-ALC APG-70 Software Development Facility (SDF)

functions.

The support of Gunship radar OFP software development can benefit significantly from direct application of the CDF, because of the use of common target processors and a common software development toolset. These development tools, including Jovial and Assembly language processors, linkers, build tools and a configuration management system, are supported in the F-15 APG-70 SDF by the CDF programming environment. This toolset is the same one used during original contractor development of the APQ-180. The impact on CDF requirements is minimal and appears to be limited to the addition of build execs for the Gunship OFP and additional storage capacity for Gunship files.

The IDRAS of the F-15 SDF, can also be applied directly to the support of the APQ-180 radar. The IDRAS, in a manner similar to the CDF, incorporates the common capabilities used in the original development and test of the Gunship radar. The only impact on the IDRAS requirements appears to be the addition of increased storage capacity for on-line file storage.

The radar system test and integration facilities, however, are potentially impacted more significantly by differences in the radars and weapon systems environments. Figure 3 illustrates the basic components of support required for radar system test. The radar test bench systems utilize the radar LRU's for full system integration, as well as other selected LRU's of the avionics system for full fidelity testing of system interfaces. Differences in the LRU sets for the Gunship, including a new Mission Computer, INS, displays and controls, as well as radar LRUs, are a significant driver in the required upgrades to an ASB or 70RTBS. The impacted areas are

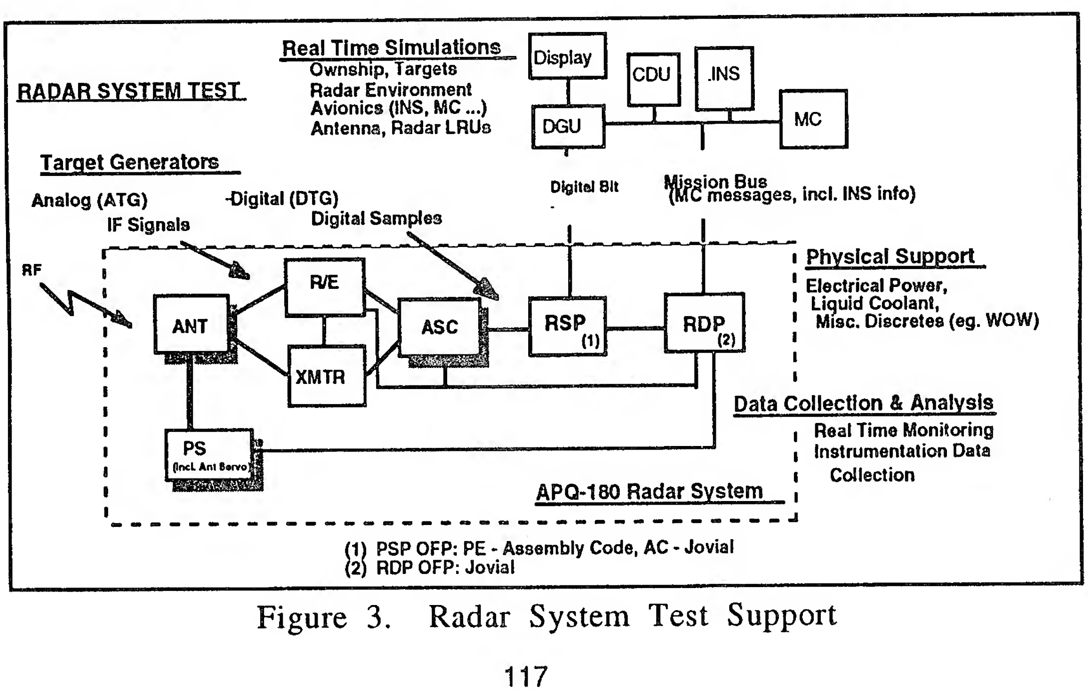


Figure 3. Radar System Test Support

primarily physical support and real time simulations.

Physical support impacts (physical fixtures, electrical power, coolant, and miscellaneous discretes) are driven primarily by the requirements of the non-radar avionics. Physical support of the radar LRU's is minimized by the use of enclosures and connectors that are common with the F-15 radar LRUs.

The real-time simulation models used in the F-15 SDF for the ownship, targets, radar environment, avionics, and radar front-end effects are based on mature simulations that have supported programs in addition to the F-15 (F-14, F-18, and B-2). Many of the changes in this area are associated with the specific format of the radar's interfaces to the rest of the Gunship's avionics.

In addition, experience gained during the development of the Gunship APQ-180 radar has focused attention on improving the laboratory support system capabilities for testing air-to-ground performance of the sensor. A significant improvement in the reduction of developmental flight test needs, and better overall cost effectiveness of system support, can be gained through target generator improvements. Endoclutter processing of ground targets and high resolution map processing test requirements indicate target generator improvements should be incorporated to improve clutter models as well as to increase the fidelity of returns for map mode support. Upgrading to full fidelity playback of recorded flight tapes is also viewed as a significant benefit, especially considering the limited fleet and constraints on flight test assets. This capability for laboratory playback would also require associated aircraft instrumentation system improvements for sensor data collection.

AF Expectations and Needs

In assessing approaches to supporting the APQ-180 radar, key considerations involve understanding the maintenance processes and their potential application to a shared facility, organizational issues, and the support strategies for the full weapon system.

Figure 4 provides an overview of the expected SOF block cycle update processes. The typical SOF block cycle update for routine system software changes is 24 months, though expectations for the Gunship have varied to as short as 12 month cycles. The complexity and frequency of block cycle updates influence support facility utilization and capacity requirements. This is a key concern in sharing a facility between distinct System Program Offices. While some high level estimates of potential utilization requirements have been made (e.g., the APQ-180 OFP is approximately half as complex

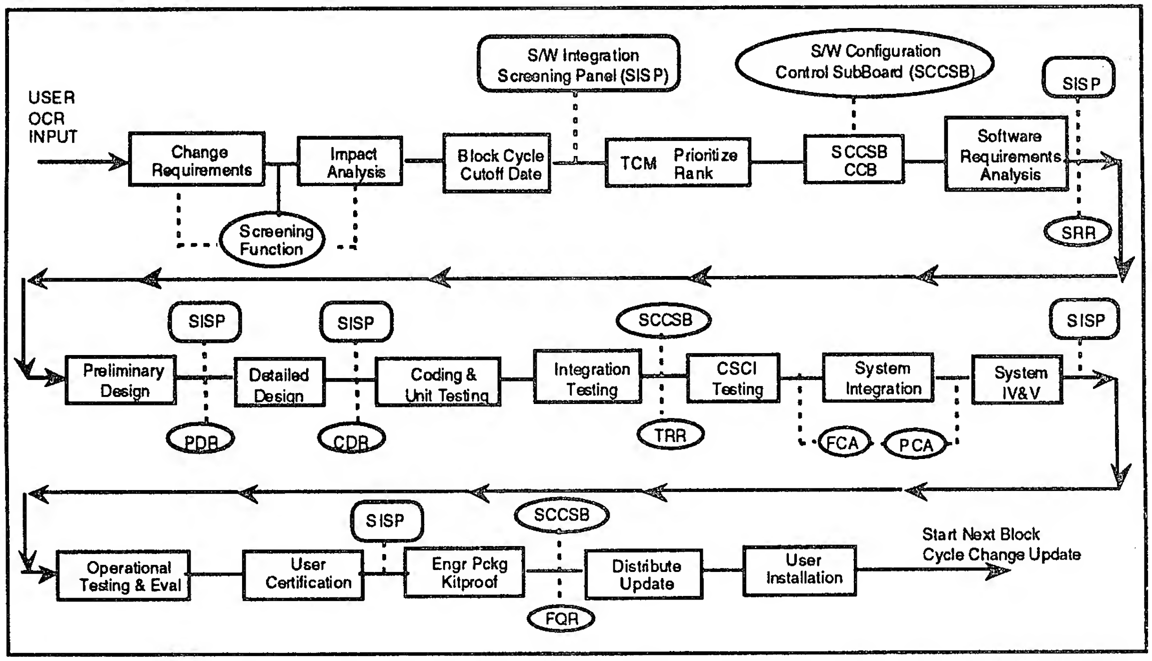


Figure 4. SOF Block Cycle Update Processes

as the APG-70 OFP, and therefore may require a 50% increase in support systems demand), the operational policies and processes regarding the use of a shared facility still need to be worked out by the government. Similarly, support personnel requirements for both the OFP maintenance activity and the support facility itself have great potential for asset leveraging (e.g., expertise and skill sets), but this issue needs to be resolved by the System Program Offices.

The support requirements of the full weapon system must also be considered in evaluating the support requirements of the radar. Integration and test of the full avionics suite, as well as operational test & evaluation, and strategies for Weapon System Integration Lab (WSIL) support versus flight test, pose trade-offs that can influence the scope of testing performed in the radar support facility. Table 4 outlines top-level strategies currently under consideration and preliminary trade-offs that have already been identified.

Preliminary technical and cost trade-offs appear to favor support strategy #1, for shared use of the F-15 APG-70 SDF for support of the APQ-180. Ease of reconfiguration of the radar system test bench and system availability are key technical factors in this support approach.

Table 4. Top Level Support Strategies and Trade-Offs

	Support Strategy	Trade-Offs
1.	Radar OFP maintenance at WR-ALC using the F-15 SDF S/W development tools, and sharing a radar test bench modified to support the APQ-180. Once the radar tape is checked out in the SDF, it would be released for operational testing in the WSIL or OT&E	- Shared F-15 facilities, with test bench APQ-180 upgrade. - Shared use F-15 common LRU's, add unique Ant, ASC, P/S - Upgrade target generators for improved A/G support - Potential leveraging of LFE APG-70 expertise. - No sharing of WSIL assets (i.e.. need separate MC, DGU, BMUX sim)
2.	Radar OFP maintenance at WR-ALC using the F-15 SDF S/W development tools, and a new dedicated radar test bench modified to support the APQ-180. Avoid conflicts on radar test bench demand.	- Shared F-15 CDF & IDRAS facilities, duplication of APG-70 radar test bench (monitors, instrumentation, simulations, etc.), with upgrades for APQ-180. - Full radar LRU set needed - Upgrade target generators for improved A/G support - Potential leveraging of LFE APG-70 expertise. - No sharing of WSIL assets (i.e.. need separate MC, DGU, BMUX sim)
3.	Use radar OFP development and analysis tools in the F-15 SDF (i.e.. CDF & IDRAS). Co-locate WSIL at WR-ALC to allow optional connection of a new radar test bench in a full avionics system configuration, as well as a stand-alone configuration for radar subsystem checkout. Fully instrumented radar test bench for radar OFP development and maintenance, shared with avionics system test. Site independent option would need a duplication of CDF & IDRAS for Gunship.	- Shared F-15 CDF & IDRAS facilities, duplication of APG-70 radar test bench (monitors, instrumentation, simulations, etc.), with upgrades for APQ-180. - Full radar LRU set needed - Upgrade target generators for improved A/G support - Potential leveraging of LFE APG-70 expertise if co-located - Sharing of WSIL assets (i.e., MC, DGU, BMUX sim) assuming expected demand is low and any concurrent radar testing is done using MC simulation.
4.	Contractor maintenance & support of the APQ-180. Currently considered a fallback option.	- Some leveraging of existing APG-70 test assets and personnel - Upgrade target generators for improved A/G support

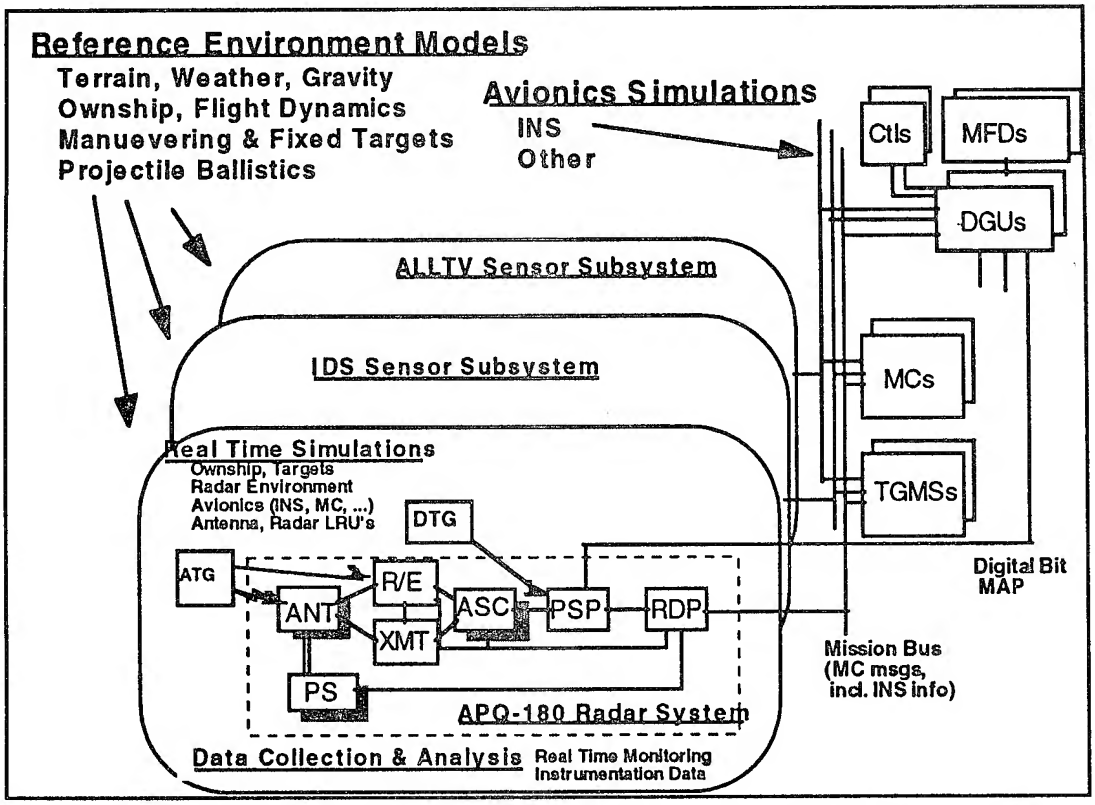


Figure 5. Radar Test Bench in a Weapons System Integration Lab Environment

The adoption of strategy #3, another favored approach, introduces technical considerations for integration of the radar test bench system into the WSIL and the coordination of the ownship and environment models with sensor specific models and timelines. Figure 5 illustrates a potential approach to the use of a radar test bench in a full avionics system test environment. This is considered reasonable from a technical viewpoint (previous versions of the radar test bench system have been used in this fashion), however this does introduce additional engineering effort into the development of the Gunship test and integration support facility. Support activity loading considerations also influence resource needs and the determination of whether a single radar test bench system and LRU set is sufficient for supporting both the development and subsystem test of the radar OFP, as well as the support of test activities associated with operational test & evaluation of the full avionics suite.

Status of Study Recommendations

Basic radar support system requirements are well understood at this time, although top level strategies adopted by the government will influence the full scope of technical requirements and the selection of an implementation approach. CDF and IDRAS capabilities are directly applicable to support of the APQ-180 radar, although additional storage capacity is required to meet APQ-180 needs. Radar test bench system modifications are also well understood, although alternatives need to be resolved based on the accessibility of the support resources of the F-15 SDF and the strategy for the Gunship WSIL.

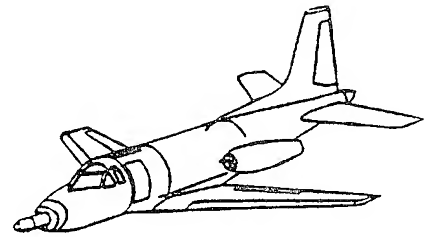
Some of the target generator and instrumentation system improvements need further refinement, and are recommended for further study. In addition, investigations of future Gunship flight instrumentation strategies are on-going and will need to be considered in the proposal for radar sensor data instrumentation.

Based on a technical assessment, sharing of the F-15 APG-70 SDF support systems appears to provide an attractive, cost effective approach to support of the APQ-180. Support process issues and work load expectations, however, still need to be resolved before such an approach or an alternative can be adopted.

References:

1. "The SDF - A Multi-Purpose Radar Avionics Software Development and Support Environment", Gary K. Miyahara, TFWGCON, 1993.
2. "Weapon System Computer Resources Management", LU Operating Instruction 63-1, SOF System Program Office, WR-ALC, 7 October 1994

WEAPONS TEST SQUADRON (WTS)
Code 561220D
Naval Air Warfare Center, Weapons Division
China Lake, CA 93555-6001
Head: Bob VanStee (619-939-5383)



Role: The Weapons Test Squadron Systems Support group, Code 561220D, develops, maintains, and operates modifications to NAWCWPNS aircraft. The Systems Support team specializes in developing and integrating modifications to the T-39 aircraft test bed for captive flight testing.

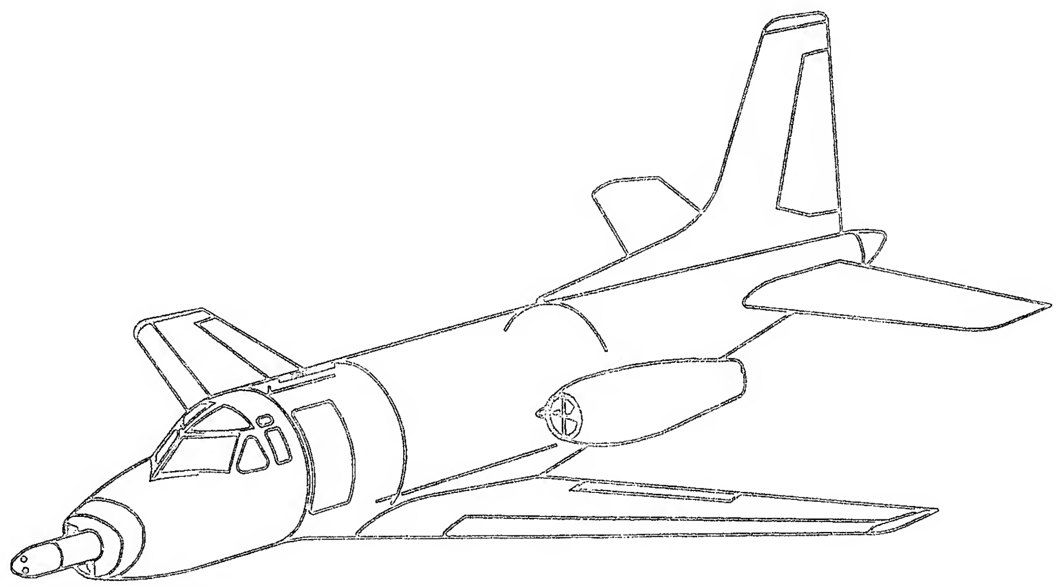
Operational Concept: The WTS supports the T-39 aircraft test bed and modifies it as necessary to support all types of captive flight testing. The T-39 is a quick-maneuvering, versatile aircraft and is ideal for evaluating various test articles. A wide variety of background clutter can be collected because of the extensive altitude range (50 - 45,000 feet), speed (120 - 350 kias) and flight duration (up to 3 hours).

Responsibilities: The Systems Support team provides engineering and integration support of systems onto aircraft for captive carry flight testing.

Facilities: The T-39 aircraft is configured with a mounting platform in the radome area that will accommodate many types of seeker, fuze or radar systems. The mounting rack is currently cleared to carry a 160-pound test article with a center of gravity 35.8 inches forward of the fuselage station 50 bulkhead. Three cameras can be mounted to this platform to provide a forward, 45° down and 45° port field of vision. The radome can house articles up to 14 inches in diameter. There is one altered radome readily available and another radome that can be altered to accommodate the modified configuration. The aft cabin of the aircraft can be arranged to fit 330 pounds of instrumentation on a rack that covers 2700 in^3 of space, while carrying 3 passengers and 2 pilots. The available power of the test bed includes Three Phase Aircraft Power (115VAC, 400A, 400Hz), 1 kW Inverter (115VAC, 8.5A, 60 Hz), 3.5 kW Converter (120VAC, 29.16A, 60 Hz) and 28 VDC (50A). Pre-routed wires from the aft cabin area to the radome include: DC Power Cables, Digital Data Cables, Digital/Analog Data lines, and AC Power Cable. Additional pre-routed cables include an Aircraft Audio Cable, DC Power Cable, and AC Power Cable.

T-39 Test Bed

Code 561220 D
Systems Support
Weapons Test Squadron, China Lake
939-5383



This document is approved for public release: Distribution is unlimited.

March 16, 1995

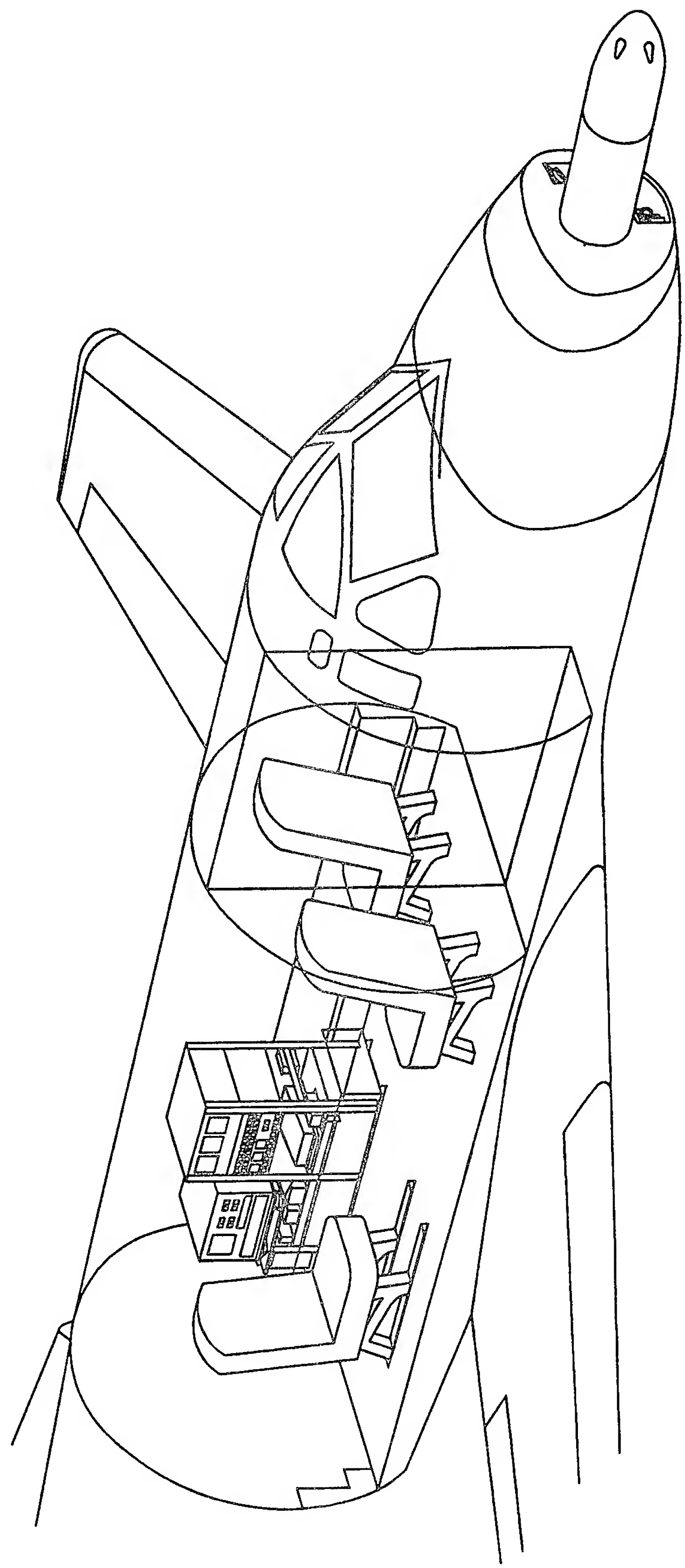


Figure 1. General Cabin Layout

T-39 Test Bed Characteristics

1.0 TEST BED PURPOSE

To provide a quick-maneuvering, versatile aircraft for captive flight testing. The T-39 is ideal for evaluating various test articles including seekers, fuzes, radar systems, etc. Four project personnel may be present onboard for data gathering and system evaluation.

2.0 AIRCRAFT

The T-39, built by North American Aviation Inc., is a low wing, twin jet aircraft. The cockpit and cabin compartments are pressurized and soundproofed for high altitude flight. Power is supplied by two Pratt and Whitney J60 gas turbine engines, located on each side of the aft fuselage. The rated sea level static thrust of each engine is 3,000 lbs at military power.

3.0 INTERIOR ARRANGEMENT

The aircraft cabin is divided into two sections: an intermediate cabin area and an aft cabin area. The intermediate cabin area contains an electronic equipment compartment and the entry door. The aft cabin area contains the flight engineer's station located next to the test hardware rack, *and three seats for extra test personnel. Seats may be removed for additional hardware. Flight personnel do not require special training or checkouts.*

4.0 FLIGHT SPECIFICATIONS

The modified T-39's performance data is listed below in Table I.

Characteristic	maximum	minimum
Speed	350 kias	120 kias
Altitude	25,000-45,000 ft.	50 feet
G's	+3.5, -1.0	-----------
Flight Time (at 250 kias and 25,000 feet altitude)	~ 3 hours	-----------
Takeoff Distance	-----------	4,500 feet
Landing Distance	-----------	6,000 feet
Max. allowable In-Flight Gross Weight	18,265 pounds	-----------
Flight Hour Cost (FY-94)	$1,370.00 per hour	

TABLE I.

The aircraft has flight capability at night and in adverse weather. However, the T-39 is prohibited from performing spins, acrobatics, and zero or negative G continuous flight for over ten seconds.

5.0 MECHANICAL INTERFACES

Test Article: The mounting rack is currently cleared to carry a 160 lb test article with a center of gravity 35.8 inches forward of the F.S. 50 (fuselage station) bulkhead. Any loading configuration creating a moment less than 5,764 lb-in, relative to F.S. 50, is acceptable. Greater loading conditions will require further stress analysis.

The test seeker is mounted to the support frame by an adapter plate. This plate can be quickly fabricated to mount a wide variety of test article sizes. However, the maximum article diameter should not exceed 14 inches.

There are three video camera mount locations on the rack. These provide a forward, 45 degree down, and 45 degree port field of vision.

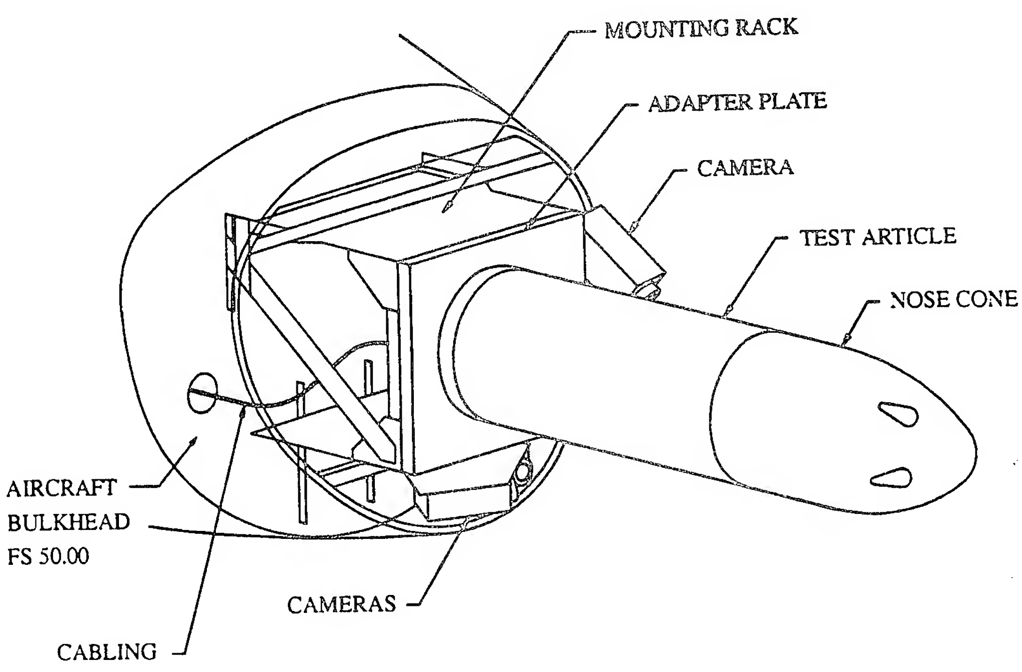


Figure 2. Test Article Mounting Arrangement

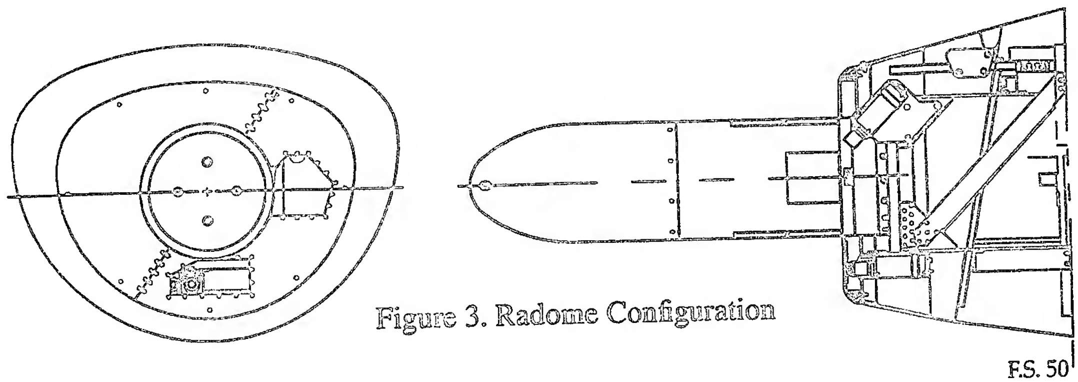


Figure 3. Radome Configuration

Instrumentation: The instrumentation rack is located in the aft cabin. Assuming the maximum onboard crew of 2 pilots and 4 passengers (180 lbs/person), 330 pounds of equipment can be mounted. There is approximately 2700 in^3 of space available for instrumentation. (Instrumentation does not need to be aircraft certified.)

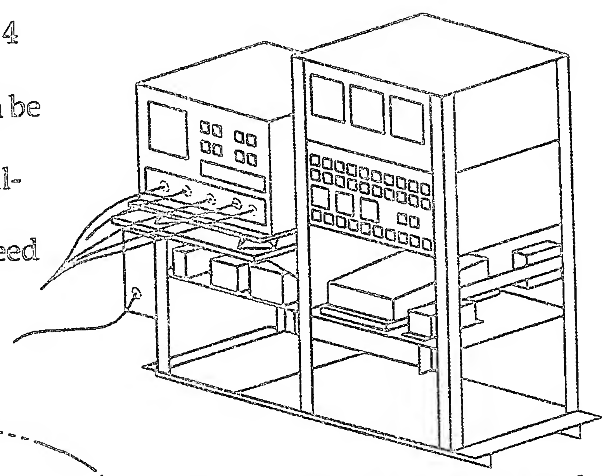


Figure 4. Instrumentation Rack

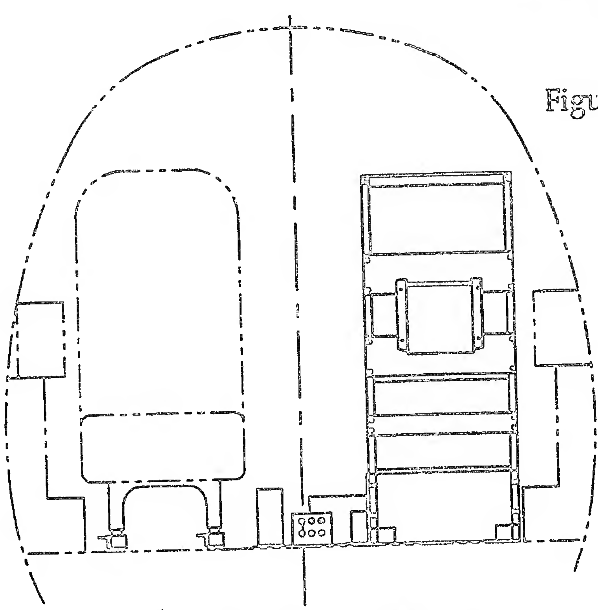


Figure 5. Aft Cabin Layout

6.0 ELECTRICAL INTERFACES

A. Available Power of Testbed

1. Three Phase Aircraft Power - 115VAC, 400A, 400Hz
2. 1 kW Inverter Supplies - 115VAC, 8.5A, 60Hz
3. 3.5kW Converter Supplies - 120 VAC, 29.16A, 60Hz
4. 28VDC, 50A

B. Available Power Supplies

The following power supplies are available for project use:

1. Rectifier - Chatham Electronics - Part # MS28123-1
   Input: 3 phase 115VAC, 5A, 400Hz
   Output: 28VDC, 55A
2. Static Inverter - Filetronics - Part # PC-15A
   Input: 28VDC, 13.2A
   Output: 115VAC, 2.17A, 400Hz
   26VAC, 2.9A, 400Hz
3. Static Inverter - Filetronics - Part # PC-15BC
   Input: 28VDC, 13.2A
   Output: 115VAC, 2.17A, 400Hz
   26VAC, 2.9A, 400Hz
4. Static Inverter - Filetronics - Part # PC-16
   Input: 28VDC, 14A
   Output: 115VAC, 2.17A, 60Hz
5. +5VDC Power Supply - CE Systems - Part # A-1046 (Quantity=2)
   Input: 115VAC, 47-420Hz
   Output: 5VDC, 2A
6. ±15VDC Power Supply - Abbot Transistor - Part # 13227
   Input: 105-125VAC, 400Hz
   Output: +15VDC, 2A
   -15VDC, 2A
7. ±15VDC Power Supply - Abbot Transistor - Part # 13671
   Input: 105-125VAC, 400Hz
   Output: +15VDC, 1A
   -15VDC, 1A
8. 28VDC Power Supply - Lambda Electronics - Part # LM2237
   Input: 110VAC, 60Hz
   Output: 28VDC, 3-5A
9. 28VDC Power Supply - PMC - Part # PXS0028V
   Input: 110VAC, 60Hz
   Output: 28VDC, 3-5A
10. 28VDC Power Supply - Transpac - Part # SR282
    Input: 115VAC, 50-400Hz
    Output: 28VDC, 2A

11. Variable DC Output Power Supply - Arnold Magnetics - Part # TV-2042 (Quantity=2)
    Input: 105-125VAC, 47-500Hz
    Output: 15-49VDC, 8-12A

C. Miscellaneous Available Equipment

1. Internal Navigation System (INS) Controller Indicator - Collins Avionics - Part # C11020/ASN
2. Time Code Generator (TCG) Programmer - mfctr.=loc. - Part # TCG-3
3. TCG Remote Control - Datum - Part # Moel 19550
4. TCG Battery Backup Unit - mfctr.=local
5. Camera Control Unit - Controls up to 3 Cameras- mfctr.=local Part # TCG-C3
6. Radar Altimeter - Honeywell - Part # 1700474-801

D. Electrical Cable Description

The cable routing in the test bed is shown in the Figure 6.

1. 1W1 and 1W2 - DC Power Cable - These two cables connect together at the bulkhead. This cable uses a cannon plug (MS3450W28-20S) at the test article to carry DC power between the test article and the equipment rack where it ends in a cannon plug (MS3456W28-20P). At present, the cable is set up for:

   Four 12 AWG +5VDC lines
   Four 12 AWG +5VDC Return lines
   One 12 AWG +28VDC line
   One 12 AWG +28VDC Return line
   Two 16 AWG +12VDC lines
   Two 16 AWG +12VDC Return lines
   Two 16 AWG -12VDC lines
   Two 16 AWG -12VDC Return lines

2. 2W1 - Digital Data Cable - This cable contains 26 twisted-shielded pairs and after being replaced by an aircraft approved cable, will be able to support 1553 Bus data rates of 1 Mbit/second. This cable uses a Deutsch connector (DS07-61S) to carry digital data from the test article to the test equipment rack where it splits into three connectors (one M24308/1-3 and two M24308/3-2).

3. 3W1 - Digital Data Cable - This cable contains 26 twisted-shielded pairs and after being replaced by an aircraft approved cable, will be able to support 1553 Bus data rates of 1 Mbit/second. This cable uses a Deutsch connector (DS07-61SX) to carry digital data from the test article to the test equipment rack where it ends in a Deutsch connector (DS07-61PX).

4. 4W1 - Digital/Analog Data Lines - This cable contains 15 coaxial cable data lines with 50Ω impedance. The cable uses a Deutsch connector (DS07-61SY) to carry data from the test article to the equipment rack where the cable splits into 26 BNC connectors (M55339/13-00492).

5. 6W1 - Digital/Analog Data Lines - This cable contains 15 coaxial cable data lines with 50Ω impedance. The cable uses 15 BNC connectors (M55339/13-00492) to carry data from the test article to the equipment rack where it terminates with 15 BNC connectors (M55339/13-00492).

6. 8W1 - Aircraft Audio Cable - This cable is a 50Ω impedance coaxial cable which uses an aircraft audio connector (PT06A-10-6P(SR)) to carry aircraft audio signals from the aircraft audio plug to the equipment rack where it ends with a BNC connector (M55339/13-00492).

7. 20W1 - AC Power Cable - This cable uses terminal lug connectors to connect AC power from the inverter/converter rack to the test article. At present, the cable is set up for:

   One 16 AWG +120VAC line
   One 16 AWG +120VAC Return line
   One 16 AWG Ground line

8. 21W1 - DC Power Cable - This cable uses a connector (MS3106A24-11S) at the 28VDC power distribution box to carry 28VDC with a maximum available current of 50A to the 1 kW inverter, where the cable terminates with the connector (MS3106A24-1P). The cable also carries 28VDC with a maximum available current of 25A to the test equipment rack which uses a connector (MS3106A22-1P). If necessary, the 25A circuit breaker may be increased to a 50A circuit breaker, thus providing the test equipment with 50A of 28VDC.

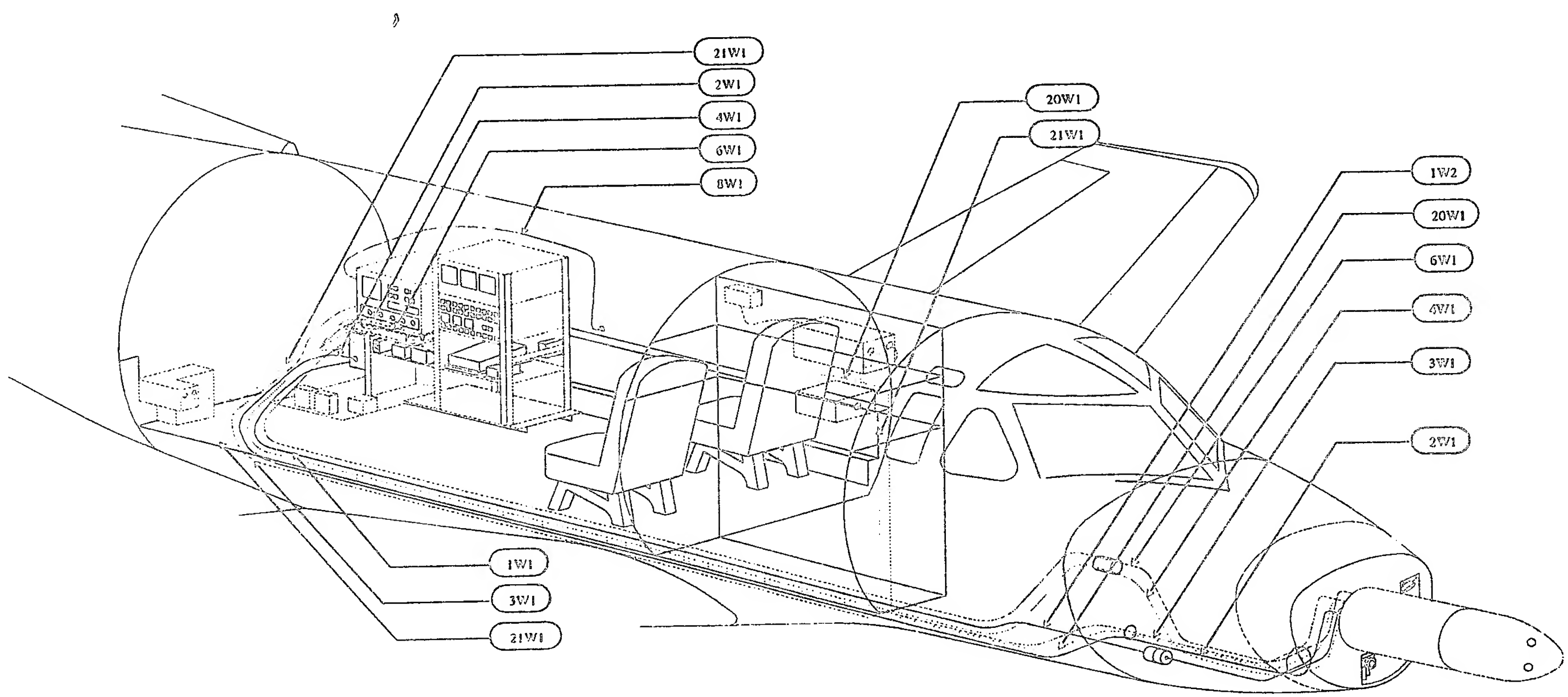


Figure 6. Cable Routing

DESIGN SUB-WORKING GROUP (DSWG)

Co-Chairs:
Robert Barry and Douglas Nester

Optical Bench for AH-1W Night Targeting System Test and Evaluation

by
Dale Billings
Darrell Grandjean
Joe Nissley
Doug Ricks
EO Section, CODE 455510D

APRIL 1995

Naval Air Warfare Center Weapons Division
China Lake, California

I. INTRODUCTION

The Night Targeting System (NTS) provides a night fighting capability for the United States Marine Corps AH-1W helicopter. The NTS system includes a chin-mounted turret (see figure 1) which contains electro-optical (EO) assemblies (a visible and thermal imager or FLIR. and a laser designator/rangefinder).

The authors designed an Optical Bench, composed of a Turret Cart and an Optics Table. This bench provides stimulus and measurement capability to verify performance of the NTS electro-optical assemblies in a laboratory environment, especially the FLIR and laser. With the Turret Cart we can transport the NTS turret from the aircraft to the Optics Table, accurately position and align it to an optical collimator in the Optics Table, measure the desired parameters, and return the turret assembly to the aircraft, with minimal disruption of flight test activity. The Optical Bench must be economical: to build, to maintain, to calibrate, to use, and of the space it occupies.

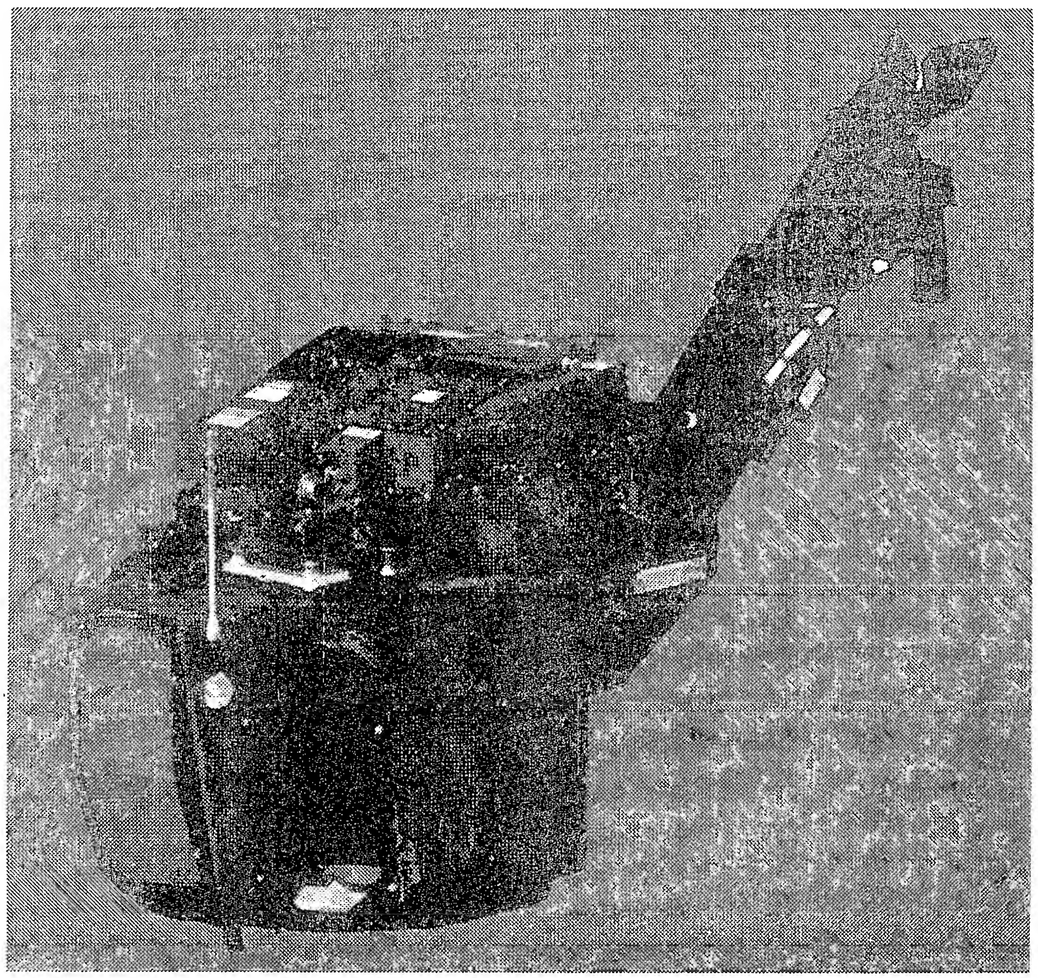


Figure 1. Kollsman Night Targeting System

The Optics Table presents images that appear to be at least several kilometers away. The actual targets are at the focus of the collimating mirror, the heart of the Optics Table. Using the mirror and targets the Optical Bench can perform much of the Acceptance Test Procedures. FLIR parameters include: square wave frequency response, dynamic range, dead and noisy channels, field of view, minimum resolvable temperature difference, geometric distortion, focusing, reticle centering, field-of-view change time, and field-of-view alignment. We can test video tracker performance a) on moving targets, b) with temporary or partial target obscuration, and c) for transition times between tracker modes.

With our boresight target and precision periscope, we can measure FLIR-to-laser boresight error and laser beam divergence. Parameters measured within our laser safety enclosure also include total energy, pulse width, sidelobe energy, stabilization time, temporal stability, coding, missing pulses, and energy stability. First, we will describe the mechanical aspects. Next we will discuss the compact, eye-safe Optics Table design, describe the associated instrumentation, and detail the example of boresight testing. Lastly, we will discuss possible improvements.

II. MECHANICAL DESIGN

The contractor's test facility for the NTS was a large room with several optical tables full of equipment, with laser safety features such as power interruption when the doors are opened. Our space is constrained, since we wished to operate the NTS turrets in the AH-1W Weapons Software Support Facility (WSSF). Instead of using the entire room as the laser safety enclosure, we built a shroud around only the avionics and optical test equipment. Our testing mostly supports flight tests, so we need to quickly and safely transport the turrets between the airframe and the Optics Table. Moreover, boresight between the turret mounting surfaces and the sensor lines of sight need to be accurately determined. These conflicting needs were harmonized in the mechanical design.

The main structural component material for the Optical Bench is 4" by 4" by 1/4"-wall 6061-T6 aluminum tubing. The choice of aluminum versus steel was for low weight. This and other materials--mostly aluminum--were welded to form a)the Turret Cart, and b)the Optics Table (collimator). The Optics Table holds the optical test equipment, and the Turret Cart holds the avionics as it is tested. The two platforms accurately dock to one another with a three-point kinematic interface shown in figure 2.

The Turret Cart provides a stable mounting platform for the NTS turret. It rigidly docks to the Optics Table using a three-point, kinematic interface. This interface eliminates the need to readjust the cart each time it is docked. The carts' rear wheels and lower framework semi-detach and remain on the ground, relieving the interface of their weight. Releasing a detente, the cart handle slides to stow under the cart. The mount for the turret, shown in figure 3, is adjustable in roll, pitch and yaw to align the turret mount to the Optics Table. The mounting bolt interface is identical to the aircraft.

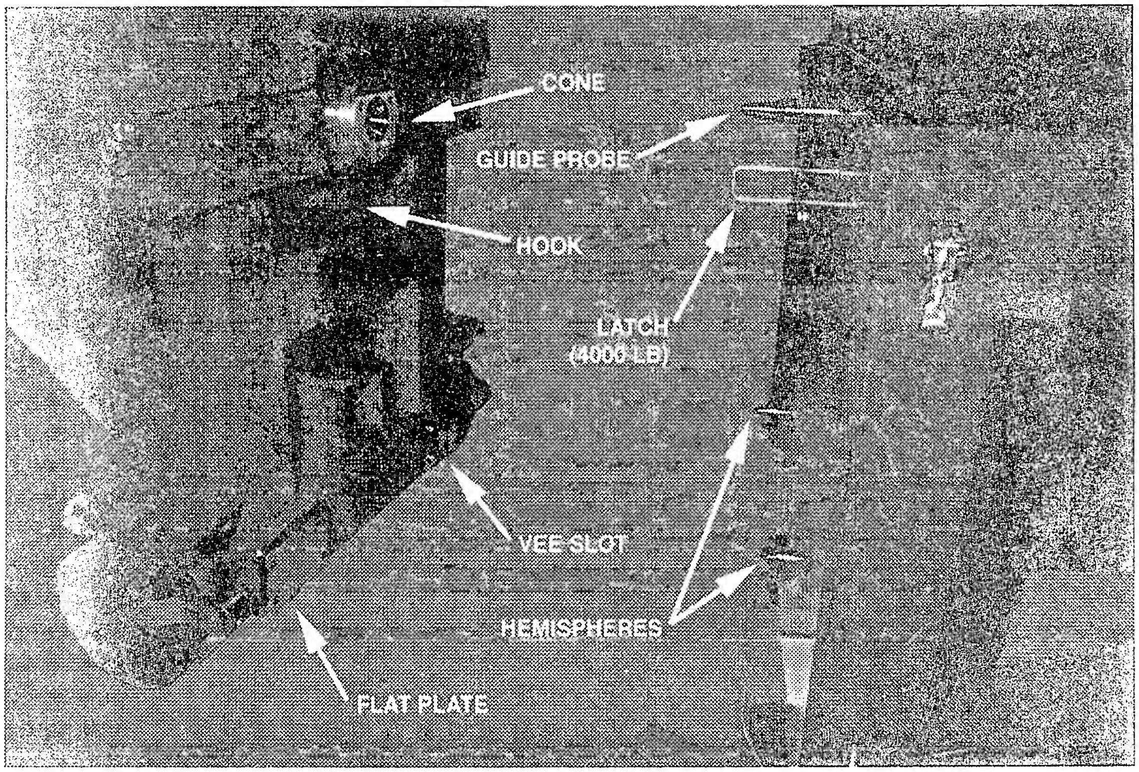


Figure 2. Kinematic Docking Interface

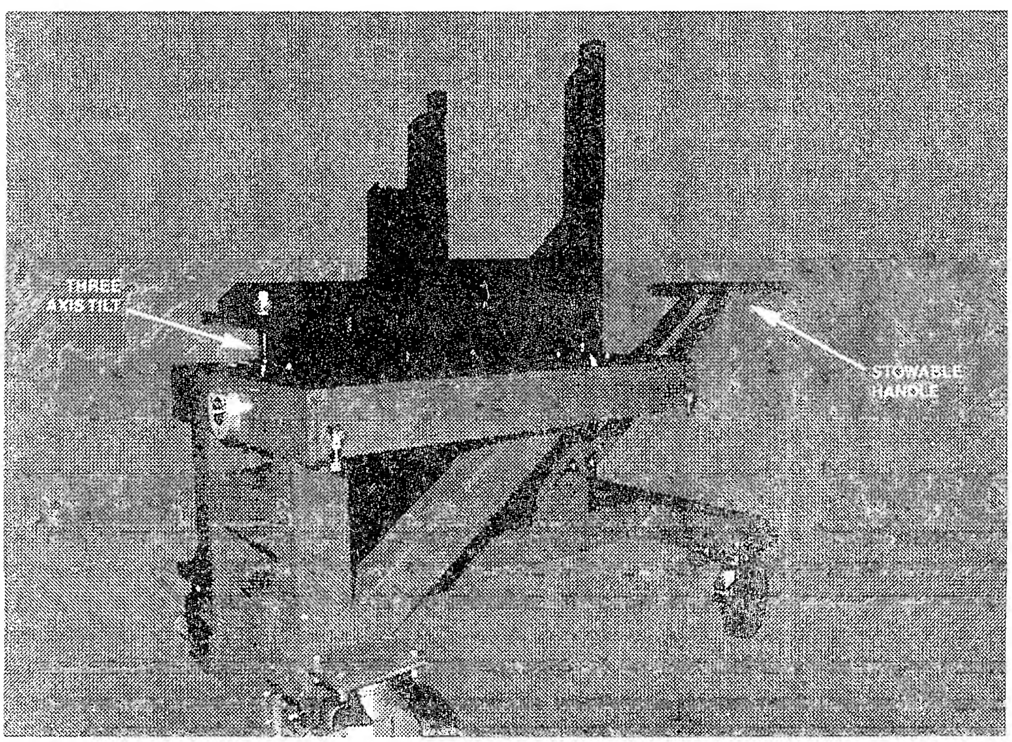


Figure 3. NTS Turret Cart

The cart rolls out to the AH-1W helicopter to transport the turret. The 7" casters accommodate rough tarmac and hangar floors. The rear casters rotate 360 degrees, while the front two are fixed forward. All four wheels have friction brakes, while the rear two lock at various angles. The rear wheels are mounted to a beam that pivots about its center for traversing surfaces that aren't flat. The weight of the cart is centered, both with and without the turret, for safety.

The Turret Cart includes a shroud (light-tight cover). The NTS turret and the test optics are enclosed by two mating shrouds to provide light-tight protection for the operator during laser tests. A pliable material and drawstring is cinched tight about the operators' eyepiece. The forward end of the cart shroud has a circular opening that mates a similar opening on the Optics Table shroud when the Turret Cart and Optics Table are in their docked position. The seal between the two shrouds is a cylindrical piece of resilient foam encompassing the 14" openings.

The Optics Table, shown in figure 4, is a tri-leg device with triangular structural features of 4" tube. This is rigid and stable with low weight. Each leg can be adjusted in height, including leveling of the Optics Table in its initial laboratory/shipboard installation. The leg adjustment is attached to a longitudinal 1/2" steel footplate that fastens the Optics Table to the floor. Each footplate has a spherical freedom of movement of 7 degrees to compensate for irregular floors. With the Optics Table stationary, a single operator can push the Turret Cart into the kinematic docking interface to couple the platforms.

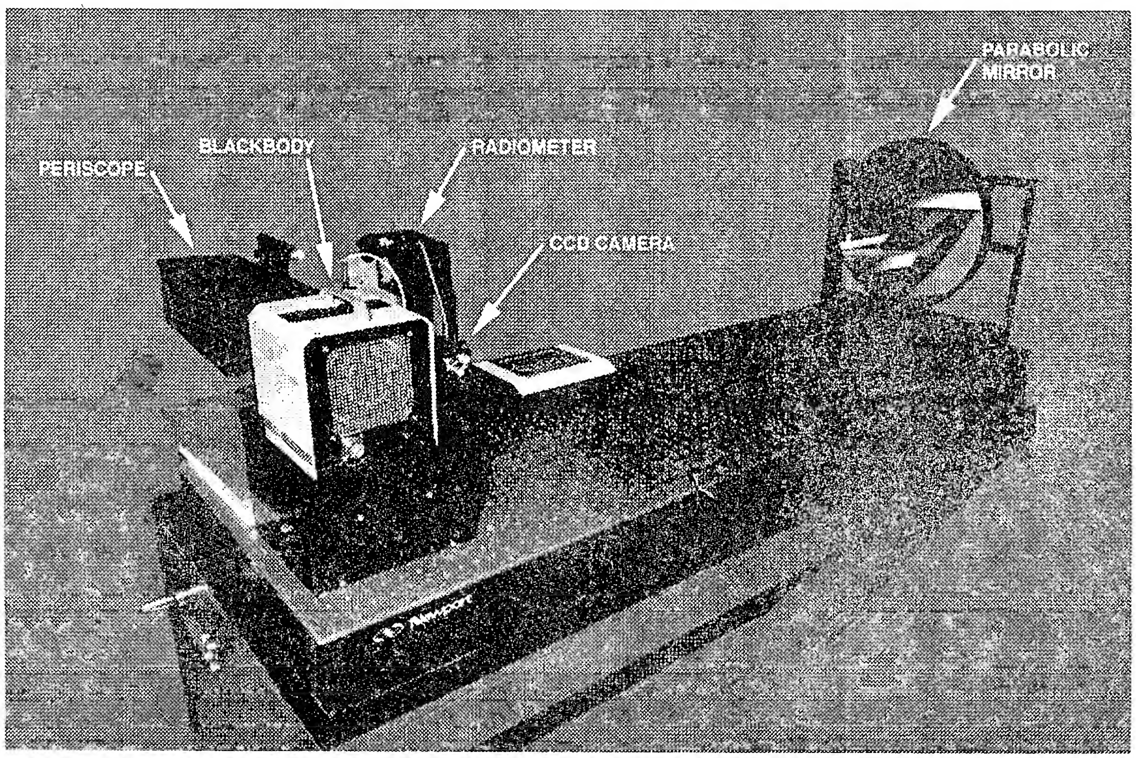


Figure 4. Optics Table

The Optics Table top is 1/2" aluminum 6061-T6 plate that has been machined to ensure perpendicularity and parallelism to its framework. The top is drilled and tapped in 5 places to accept a 2' by 6' honeycomb optical breadboard from Newport Research. Five concentrically threaded fasteners (Newport's microlock tiedowns) are used to independently level the breadboard. The breadboard is offset to one side of the table to leave room for electrical connections through the table surface. This leaves the shroud structure clear of connectors except where necessary for laser safety interlocks. The electrical connections pass through a machined port in one of the 4" by 4" aluminum square tubes, and through the 1/2" aluminum top. A sub-panel with the appropriate connectors is attached to the bottom of the square tube to provide light-tight electronic connection to the Optics Table test gear. A similar light-tight connection port is provided at the aft end of the Turret Cart to provide electronic interface from the turret to related avionics and test equipment.

The Optics Table side of the kinematic interface has a 4000lb capability cam action lever latch that can be adjusted forward and aft and operated by one person to dock/connect the Turret Cart to the Optics Table. The latch mates to a hook on the cart.

The Optics Table shroud covers the entire top of the table. The shrouds for both the Optics Table and Turret Cart are made of 1/8" 6061-T6 aluminum sheet to accommodate frequent removal and installation, and reinforced to allow test gear to rest on their top surfaces. There is a light-tight access port in the forward top side of the Optics Table shroud. This port allows the Technician/Engineer to change optical targets without removing the entire shroud. The compact, rigid, platform design will allow a)eye-safe testing in a small package, b)safe transport of the NTS turret, and c)convenient, accurate alignment between the turret mounting interface and the Optics Table.

III. OPTICS DESIGN

The sensors on the NTS turret look through the 14" circular opening in the shrouds described above. These sensors are tested with just one 8" collimator aperture. This is accomplished with a precision periscope. Figure 5 shows a simulation of the laser beam path through the Optics Table, with and without the periscope. The periscope allows a smaller aperture to be shifted to within the largest aperture, without altering the direction of the rays. The periscope relieves the tight collimator defocus tolerance associated with boresight alignments between apertures spaced even a few inches apart. The parabolic mirror also aids in these measurements since it has no wavelength dispersion. Without the dispersion associated with refractive collimators, all wavelengths focus at the same focal point. The major drawback of reflective optics is the narrow field of view.

Key trade-off factors for a collimator design include the focal length of the primary mirror, the number of expensive fold mirrors needed, the amount of floor space available, and the blackbody aperture size. If the focal length is too long, the blackbody source and floorspace become quite large. The blackbody is usually shipped away for calibration. Our design has no large fold flats, but a relatively short focal length mirror. This was largely to avoid the expense of large fold flats and large blackbodies, and their contributions to the aberration and thermal error budgets.

Using an on-axis, parabolic, 16-inch-diameter, amateur-astronomy mirror and mount saved us time and money, compared to buying a smaller (8-inch) off-axis mirror.

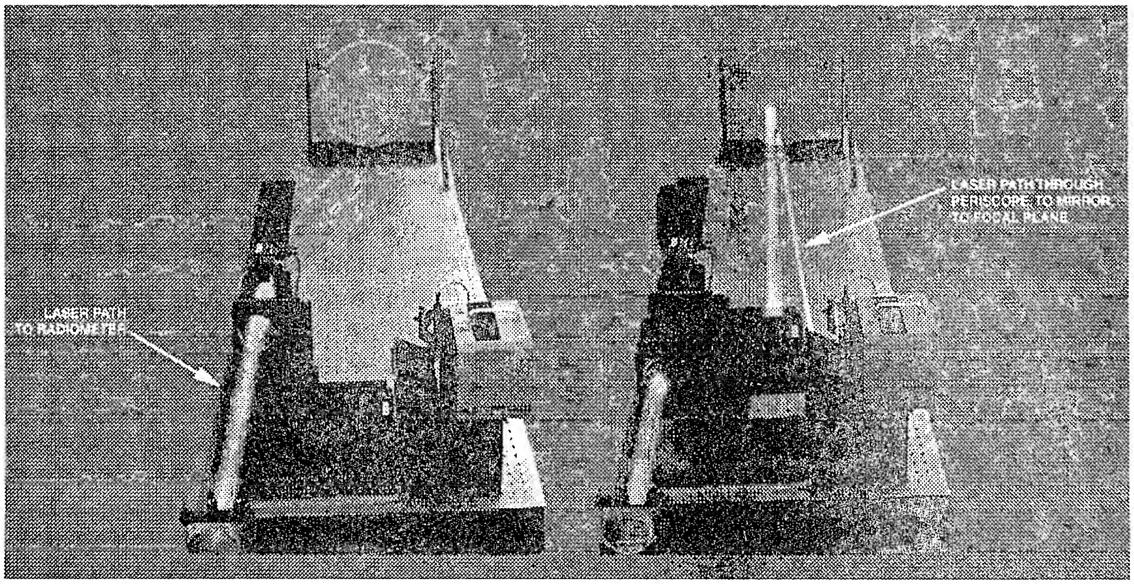


Figure 5. Laser Optical Paths

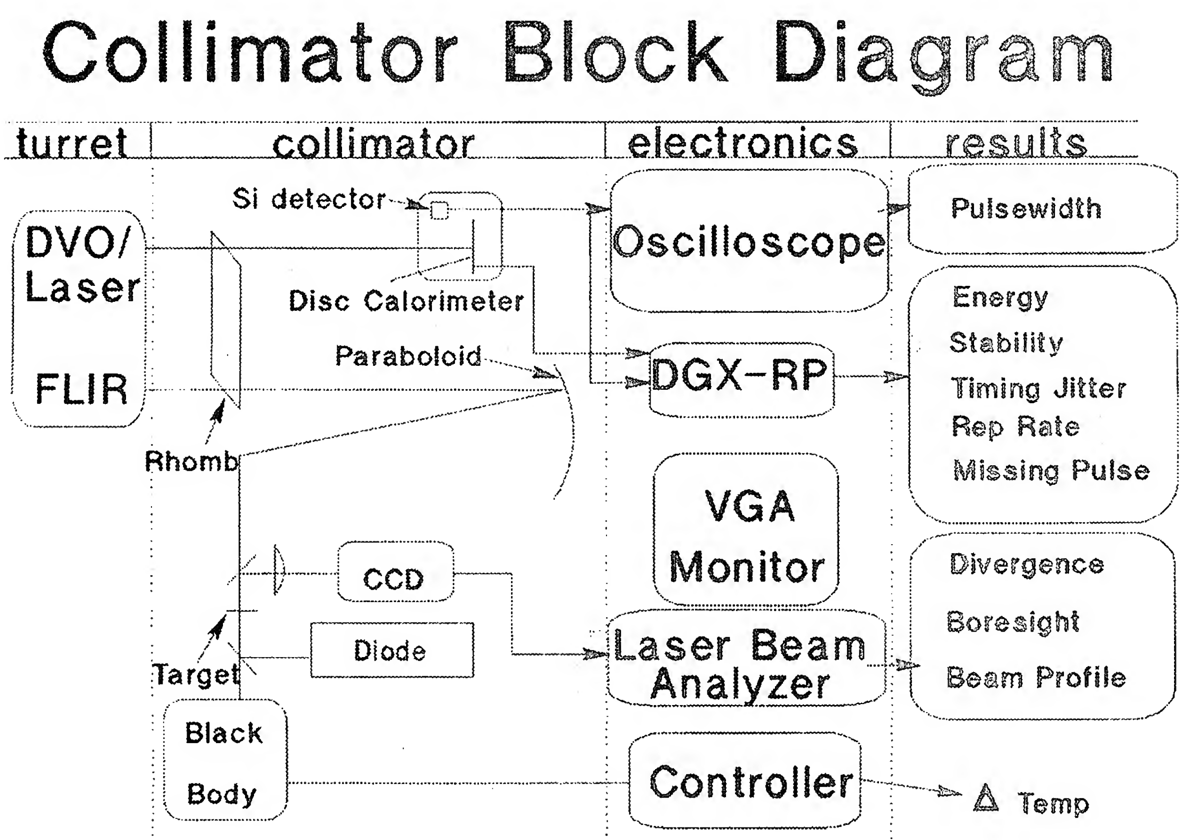


Figure 6. Collimator Block Diagram

IV INSTRUMENTATION AND CALIBRATION

A block diagram of the Optical Bench is shown in Figure 6. Laser measurements use the radiometer or the laser beam analyzer, while FLIR tests use the blackbody.

The radiometer tests the following laser parameters: pulsewidth, stabilization time, temporal stability, coding, missing pulses, energy stability, and output pulse energy. We prefer calorimetric radiometers which feature calibration by an electrical substitution heater. A standard voltmeter thereby becomes the immediate calibration reference. The slow thermopile detectors used in most calorimeters must be supplemented with a fast silicon photodiode to support our many test requirements. The model DGX-RP from Ophir was found to provide all of these capabilities in one compact radiometer with an RS-232 output.

The laser beam analyzer tests beam divergence, sidelobes (part of beam profile), and FLIR-to-laser boresight. The CCD camera should be selected for wide dynamic range and compatibility with the selected beam analyzer.

Commercial blackbody sources have short calibration intervals. Thus a vendor with nearby calibration facilities is helpful. The tests performed using the blackbody include: Square Wave Frequency Response(SWFR), dynamic range, dead/noisy channels, field of view, Minimum Resolvable Temperature Difference (MRTD), distortion, focusing range, moving target track, predictor with obscuration, transition times between modes and fields of view, and reticle centering. Tracking tests use a small pinhole target which travels around a circle of two degree diameter in the turrets' field of regard (see Figure 7). The motor used for this test is a simple synchronous clock motor for accurate rotation rate. This limits the tests to a single speed, but costs less than variable drives.

All other FLIR tests use simple target masks such as shown in figure 8. We decided to use a blackbody target format only four inches square. This requires the interchanging of targets, as opposed to using an array of targets. However, reducing the collimator field of view gives us more confidence in the target thermal uniformity and helps reduce the collimator aberrations.

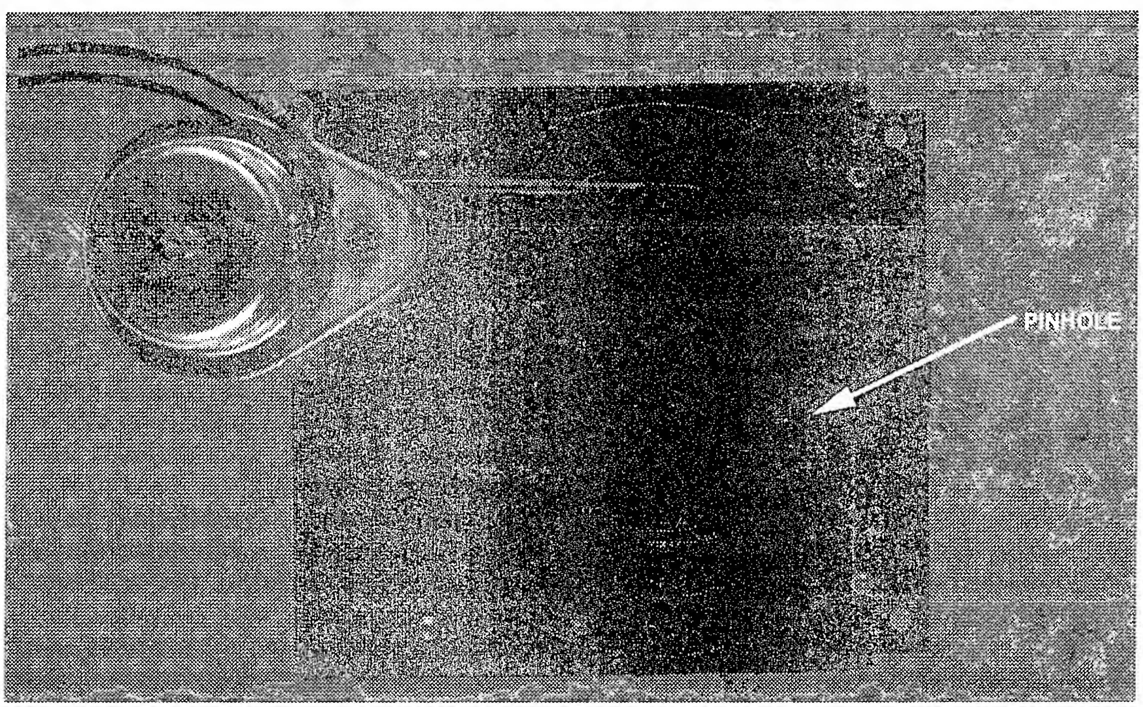


Figure 7. Rotating Target

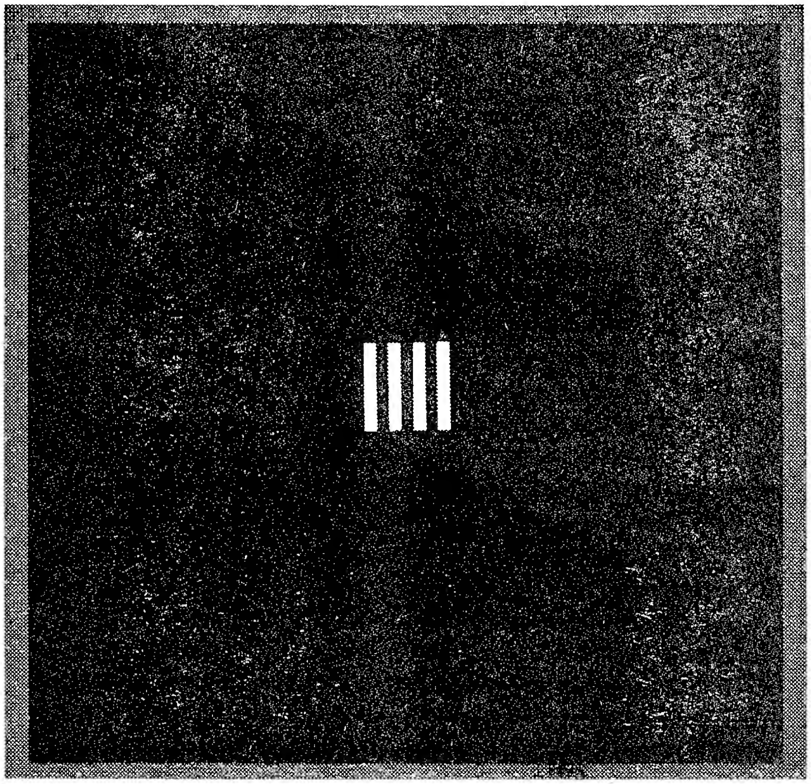


Figure 8. Simple Target Mask

V. EXAMPLE: BORESIGHT

As an example, we will describe the FLIR-to-laser boresight test. First, we insert the periscope so it shifts the laser aperture into the middle of the FLIR aperture, which is large enough to see around the periscope. Then we put a square boresight target with fiducials at the focus, cool the FLIR, lock the autotracker onto the boresight target, fire the laser, and sample the laser spot with the CCD camera and laser beam analyzer.

Thus the CCD video shows the centering of the laser spot on the square target the FLIR is autotracking. Figure 9 shows the focal plane with the boresight target installed, and the CCD camera near the blackbody. The red diode laser at the top of the target plate illuminates the target. An iris adjusts for laser intensity, since the camera AGC does not follow the short pulses correctly. The laser beam analyzer processes the CCD video image to determine the offset between the laser spot image and the square target. Dividing the offset by the spacing between the fiducials, and multiplying by the known angle between the fiducials yields the angular value of the boresight error. Measuring between centers of symmetrical objects (like our fiducials) eliminates the variable of setting a threshold for determining where the edge occurs: which shade of gray is on the white side of the edge? With the fiducials, a pixel centered between edges can more accurately be called a center, so linearity of the camera/digitizer response curve is less critical.

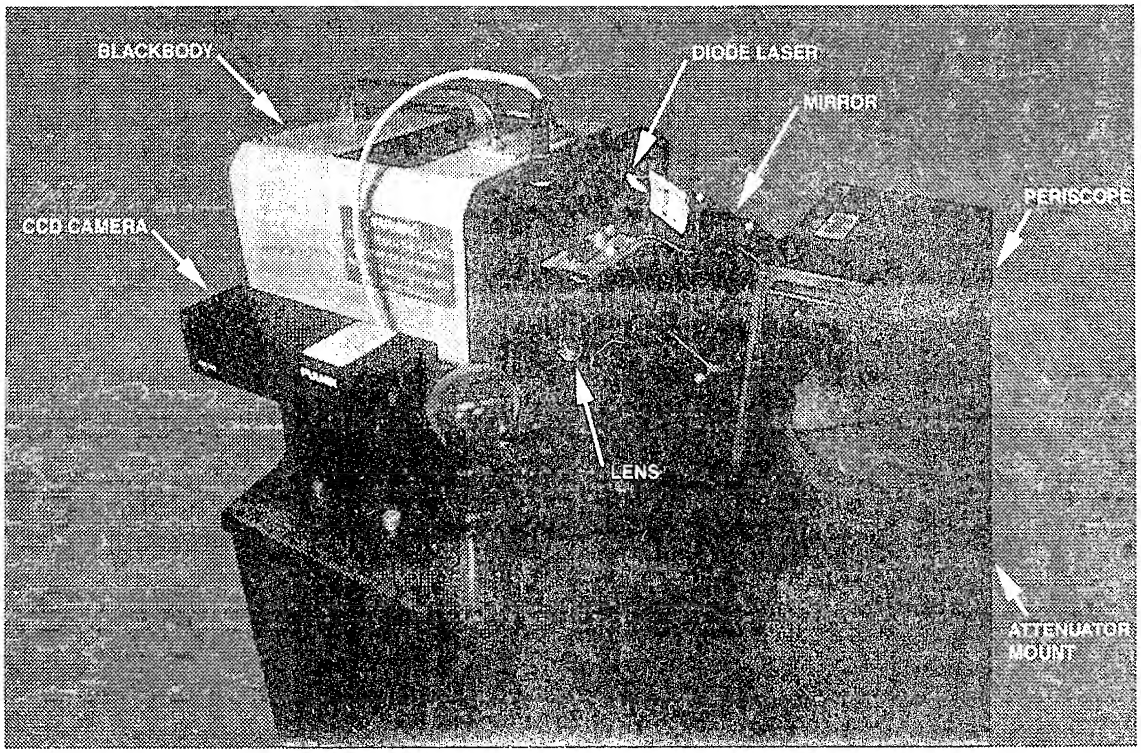


Figure 9. Boresight Target in Focal Plane

VI. FUTURE GROWTH AND CONCLUSION

Several ways to improve the Optical Bench have been suggested. First, we could support more of the ATP tests by providing the ability to test a larger field of view. This could be accomplished with the addition of a steering mirror to move the image of the test targets around in the field of view of the avionics. Another suggested improvement is an automated focus monitor to avoid requiring engineering support of the Optical Bench to periodically check the focus with the ronchi grating. This could extend diffraction-limited performance to the visible spectrum.

The Optical Bench is compact enough to be used in virtually any avionics development facility. The field of view of the collimation optics is 3 by 2 degrees (vertical/azimuth). The Optical Bench design can accommodate upgrades such as complex moving targets (in two axes), an infrared scene generator (of tanks, planes, flares, etc.), or a laser countermeasure source.

Other upgrades not included in this unit are a) a coil of optical fiber (calibrated in length) to provide a time-of-flight delay to test the laser rangefinder, and b) diode sources in the spectral band of the rangefinder to test the first and last pulse logic, or to test laser spot tracker systems.

In short the flexibility of this design can provide verification and validation of production hardware assets as well as a research and design performance characterization tool, all at a relatively modest cost.

VII. ACKNOWLEDGMENTS

We would like to thank Frank Foust for his diligent work in detailed mechanical design, and Greg Lundin and Wayne Willhite for their managerial patience. Gerald Adams was also helpful in gathering design data from the subcontractor.

NETWORK SWITCHING IN THE VIRTUAL TEST STATION (VTS)

Harold Dean
Robert Norton
Science Applications International Corporation
Dayton, Ohio, 45432

Abstract

The development of the Virtual Test Station (VTS) by the Avionics Logistics Branch of Wright Laboratories (WL/AAAF) and SAIC has spawned the creation of a new hardware system for switching the avionics and network signals that move between resources in the VTS. This new device is called the Modular Switching System (MSS) and makes use of state-of-the-art cross-point and analog switch technologies to switch a wide variety of signal types ranging from low frequency analog signals to digital signals up to 600 Mbps. The MSS is comprised of a base Modular Switching Unit (MSU) and up to 32 Modular Communication Plug-ins (MCPs). The system embodies WL/AAAF's fundamental design philosophy of modularity, flexibility, reconfigurability, maintainability, and low cost. The MSS does not represent a radical change in network switching technology. Rather, it is the integration of several existing and emerging technologies in a way that brings new flexibility and capability to Integration Support Facilities (ISFs) and other organizations with a large variety of high performance computing resources that can benefit from the ability to rapidly change configurations. This paper describes the operation and design of the MSS and how it can be applied in various switching applications.

Background

Research by WL/AAAF, under the Embedded Computer Resources Support Improvement Program (ESIP) program, has been focused on simulator architecture design concepts. The principal focus has been on the insertion of new technologies to enable the modularization of test station design. A modular approach provides the test engineer a greater degree of flexibility to manage test station resources within the ISF. The increased flexibility enhances automated avionics test and reduces the cost of test station development and maintenance. The benefits of a modular concept were successfully demonstrated by the Advanced Multi-Purpose Support Environment (AMPSE), a modular, reconfigurable test station developed by WL/AAAF.

The traditional approach to avionics support was to build a dedicated test station for each avionics subsystem and its embedded computer. A mainframe computer was an integral component of each test station. There was little or no cross utilization of avionics or computer resources on different test stations. As modern aircraft systems evolved into a distributed architecture of multiple avionics subsystems, the line replaceable unit (LRU) count significantly increased and so did the effort to support avionics test. As a result, test station development and maintenance costs soared in an effort to provide an integrated test environment. A significant part of the test station cost was the redundant sets of avionics equipment required to build multiple test stations, as well as the simulator development for each station. The principal

design goal of the AMPSE architecture was to allocate the simulation, previously hosted on a mainframe computer, across a distributed network of parallel processing resources. The rest of the test station simulation functionality was also partitioned and allocated to individual modular components in the AMPSE. Communication among the modular, distributed components was supported by the Shared Memory Architecture Real-Time Network (SMARTNet), a real-time data communications network using reflective memory. In addition, each processing element used a common, modular software executive, referred to as the Distributed Ada Real-Time Executive (DARTE). The modularity of the AMPSE technology provided the test engineer with a more efficient method of developing, testing, and maintaining avionics software. Although the AMPSE architecture is modular, avionics resources remained dedicated to specific test configurations. To fully capitalize on the modularity of the AMPSE architecture, WL/AAAF created the VTS. In the VTS, all of the communication signals from the modular resources are routed to the MSS (Figure 1) where they can be connected under remote control to form a variety of "virtual" test stations.

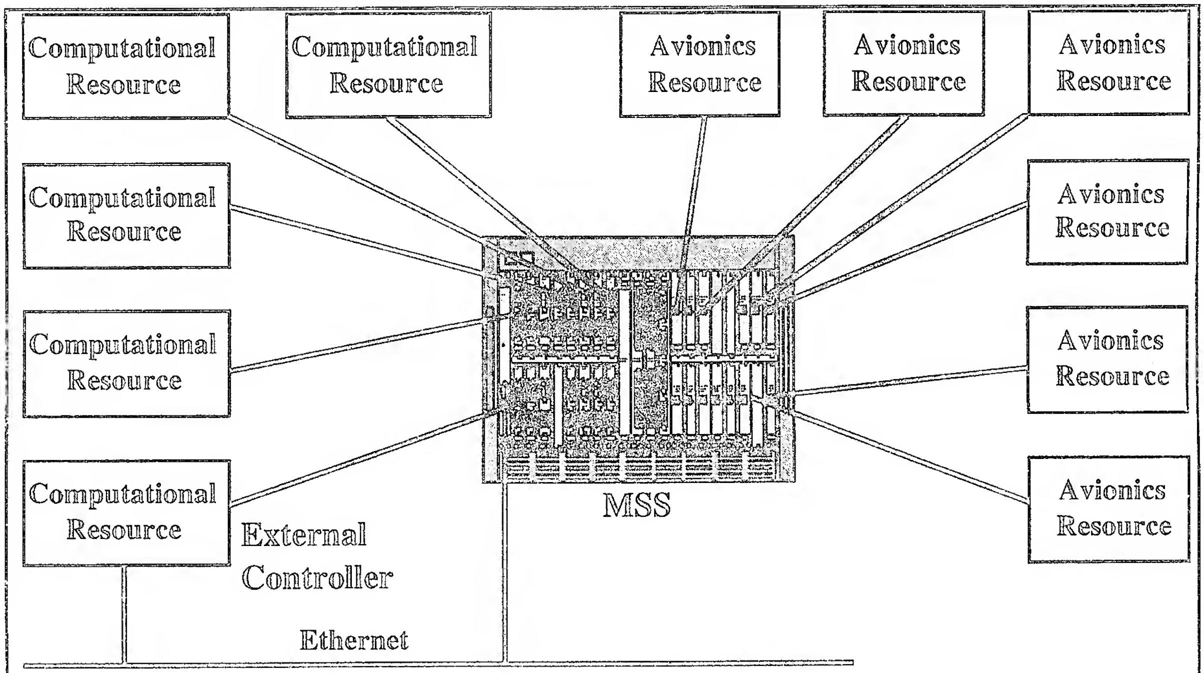


Figure 1. MSS in the VTS Concept

Concept

The MSS is a software controlled, integrated data switching system. It is composed of the Modular Switching Unit (MSU) and the Modular Communications Plug-ins (MCPs). The MSU consists of the system chassis and the switching components. The MCPs customize the system to switch a particular configuration of signals. The Modular Switching Unit with a complement of Modular Communications Plug-ins, forms the signal switching hardware component. The

MSS is designed to be remotely controlled from a workstation or other computer. Communications between the MSS and the computer directing the switching is handled over Ethernet.

The MSS hardware architecture is specifically designed to operate in a real-time simulation environment. The modular architecture supports the switching of both analog and digital signal formats. The MSS switching topologies employ what is commonly referred to as "circuit switching" versus the "packet switching" methods used in data communications. Since the MSS physically switches the transmission media, the delays associated with receiving and re-transmitting packets are avoided. The MSS design avoids the expense of custom semiconductor development by using the same semiconductor technology being applied to the emerging communications standards such as Fiber Channel, the Synchronous Optical Network (SONET), and the Asynchronous Transfer Mode (ATM). The MSS concept optimizes the sharing of resources and eliminates the costly expense of building/configuring multiple test stations. The development of the MSS is the enabling hardware technology required to implement the VTS concept.

Switch Requirements

The VTS requirements levied on the MSS are formidable. The switch must handle signal frequency bandwidths from DC to 200 Mbps and voltages from millivolts to at least 28 volts. In order to handle future data networks, frequency bandwidth growth capacity to 600 Mbps is also required.

The signal types are varied and cover all aircraft signal types and a variety of special signal types needed by the ISF. The most obvious signal type necessary for avionics system testing is MIL-STD-1553B. This is the most common aircraft data bus. The switching system must also switch any variants of the standard 1553B bus, such as the F-16 Weapons Bus. The switching system must provide switching for AC and DC analog voltages in various polarities and levels. The switching system must switch all of the common avionics discrete configurations: 28 volt/ground, open/ground, and 28 volt/open. The switch must be versatile enough to handle any avionics signals likely to be available in the near future. And last, but certainly not least, the switch must prevent interconnection of incompatible signals to prevent damage to the avionics and test equipment.

The special signals needed in the test facility are primarily related to the interconnection of the computer systems and the avionics under test. The most versatile arrangement would allow any computing resource to be flexibly connected to any avionics resource. Having flexible configurability optimizes the use of expensive resources by allowing multiple OFP engineers to run simultaneous testing if adequate resources are available. Configurability also enhances testing reliability and schedule adherence by routing around downed equipment. The primary signal types related to the computer configuration are high performance data bus protocols. These range from RS-232 at 300 baud to fiber optic reflective memory systems with signals in the 200 Mbps range. Additionally, Ethernet or Token Ring networks may require switching to support specific test requirements.

Approach

The MSS has two primary design drivers: signal frequency and signal topology. The switch design has to cover the frequency range from DC to 600 MHz. There are a variety of analog switches that cover the DC to 2 MHz range. These are medium density type components with sixteen to thirty-two switches per integrated circuit. There are high frequency analog switch components which extends the range to 200 MHz. However, these are low density circuits with one or two switches per component. RF style switches exist which can handle 600 MHz to 1.2 GHz signals, but these need to be tuned to specific bands and they are very low density components, one signal per module, with limited expansion capability. Finally, there are a variety of Emitter Coupled Logic (ECL) cross-point switches being developed for the telecommunications industry. These are high density switching components with an entire 32 x 32 switching matrix in a single integrated circuit. They handle the frequency range from DC to over 1.2 GHz. and some units provides signals to support expansion.

The cross-point switches satisfied the frequency requirements. However, the signal topology requirements were still an issue. The potential signals can be grouped into two basic, electrically incompatible, signal topologies: bi-directional multi-drop signals and unidirectional point-to-point signals. The bi-directional, multi-drop topology includes signal types like MIL-STD-1553B and Ethernet and is characterized by data that can flow in either direction on the single transfer cable. The fact that data flows in both directions on the single wire prevents any kind of simple amplification or buffering. The unidirectional, point-to-point signal topology includes signal types like the fiber optic reflective memory data signals, synchro signals, analog signals, and avionics discrete signals. This topology is characterized by data that always flows from a single source to one or more destinations. This type of signal can be easily amplified or buffered and only needs an electrical signal conversion to be compatible with the crosspoint switch component. There is no single switch component type which is ideal for both topologies. The simplest way to handle both signal types is to use both analog switches for the bi-directional signals and crosspoint switches for the uni-directional signals. The design challenge was to find a method to package both switch types in a format which was compact, easy to use, and expandable.

The design concept for the MSS is to provide a processor controlled family of switching and signal conversion modules which can be installed into the MSS chassis, in any configuration, to suit individual test facility requirements. The MSS is composed of a dedicated custom MSS chassis, a COTS single board computer, a family of switch boards, and a family of Modular Communication Plug-ins (signal personality modules). The switch boards and MCPs can be installed in various configurations to meet the specific needs of the test facility. Multiple MSS chassis can also be interconnected to form large switching arrays. The MSS firmware provides an auto-configure capability which reads the configuration of all switch and MCP modules and monitors all switch commands for safe electrical operation. The firmware ensures that only compatible signal types are interconnected.

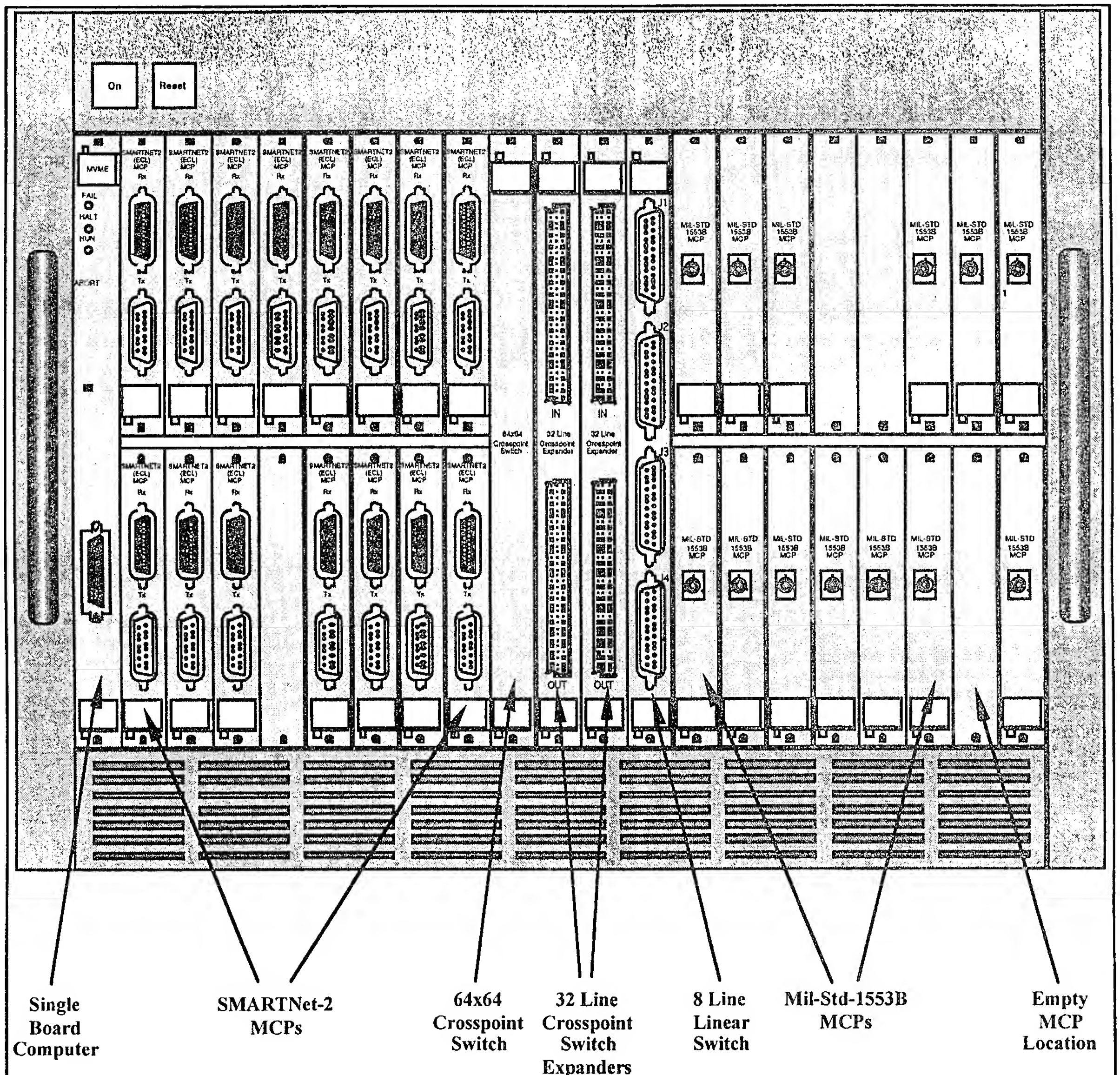


Figure 2. Modular Switching System

The MSS chassis is a nineteen inch, rack mount, Eurocard style chassis. Figure 2 shows an example of an MSS chassis with one potential card configuration. This example is configured to switch both SMARTNet-2 and MIL-STD-1553B signals. The chassis provides mounting space for one 6u COTS VME, single board computer; up to four 6u x 220mm switch modules; and up to thirty two 3u x 220mm MCP modules. The chassis can be used as a stand-alone switcher or connected with others to form large interoperable switch arrays. The chassis has a custom motherboard with slot 1 dedicated to the single board computer. Motherboard slots 2 through 9 and 14 through 21 are configured to hold two banks of 3u MCP modules. Motherboard slots 10 through 13 are configured to hold 6u switching modules. The motherboard provides the wiring to

form two, electrically isolated, signal switching arrays. Each MCP location is wired for one differential signal pair which can be used for bi-directional signal transfer to the switch array. Each MCP location is also wired for two differential output signal pairs and two differential input signal pairs which are separately wired to the switch array for uni-directional signal transfer. The fact that the interconnect wiring is permanently available in the chassis backplane vastly simplifies the switching system components. Switch modules and MCPs are designed to only connect electrically to the compatible wiring array, thus preventing electrical damage. Since all the positions are wired the same, with the exception of addressing codes, there is only one configuration constraint. The single constraint is that the system must have a switch module for any signal array that has MCPs connected. Providing wiring for both arrays produced a lot of signals in the backplane. There are almost 600 signals routed to the four 6u switch assembly positions. The switch modules were purposely located in the center of the chassis to allow signals to be routed from both sides. This improves the electrical isolation and minimizes crosstalk. The chassis provides global addressing for the MCPs and switch assemblies so there are no address jumpers to set. MCP modules get their address from the slot in which they are installed. All MCP locations provide identical capabilities.

The embedded controller is a COTS VME single board computer (SBC) with built-in Ethernet. The operating code (firmware) is stored in ROM or can be optionally downloaded from an external host computer. The SBC controls the switching system over a modified VME control bus. It accesses all switch modules and MCPs as A16, D16 VME slave modules. The SBC is the sole bus master and no interrupts are required except the VME BERR. Each switch module and MCP location has a hardwired backplane address and is required to have a status register at the base address for the board. By interrogating the status register for each location, the embedded controller determines the switch and MCP configuration for the MSS. The configuration and operating rules for all MCP and switch module types are contained in the firmware. During initialization, the firmware autoconfigures the run-time software to match the hardware configuration. From that point on, the firmware will test each connection command against the stored rules to validate legal connections. Requested connections which violate the rules are reported back to the external system directing the MSS. When multiple chassis are inter-connected by the expansion ports, the firmware from each unit communicates with all connected units to coordinate and test inter-unit switching. The user only needs to specify connections by unit number and channel to any unit in the array and the connection will be made.

The switching modules provide controlled switching of selected signals. Since bi-directional signals appear on physically separate pins in the 6u connector from the single ended signal types, switch modules are designed to optimally switch individual types. Since all signals are available at each location, there are no constraints on switch module locations. All switch modules are designed with a status register located at the boards base address. This status register contains a type code field which is assigned to each switch design. By reading the status register and decoding the type code, the firmware can recognize and control each switch module in any of the four locations.

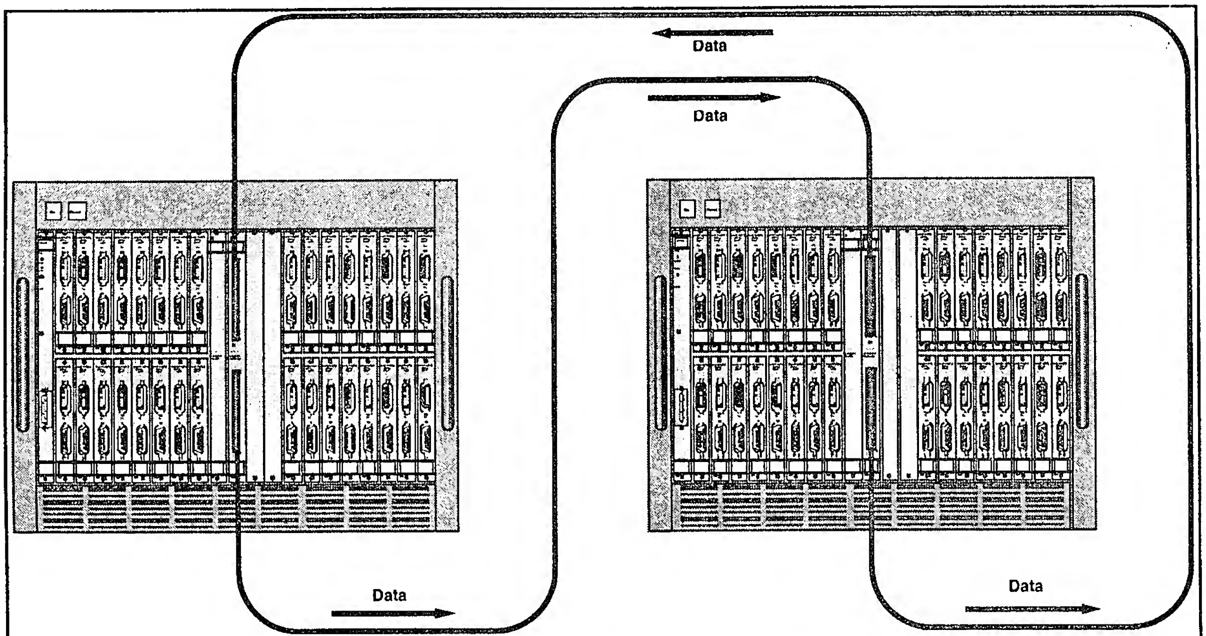


Figure 3. MSS Crosspoint Expansion Example

There are currently three switch modules defined: a 64x64 crosspoint switch module for switching single ended signals, a 32 line crosspoint expansion module for passing single ended signals between MSS chassis for switch array expansion, and an 8 line linear switch module for switching bi-directional signals. The 64x64 crosspoint switch module is electrically connected to the single ended signals from the chassis motherboard. This one module in a chassis with thirty-two single ended MCP modules provides all the switching capacity needed to connect any available input to any available output. The 32 line crosspoint expander connects multiple MSS chassis together (Figure 3) such that any of the single ended inputs in any chassis can be connected to the single ended outputs in any other chassis. The expander board provides for the selection of up to thirty-two signals from the sixty-four available signals in a chassis. The 32 line expanders are used to connect the chassis in a ring topology network. Depending upon cable length, from six to eight chassis may be connected into a single ring. If more than thirty-two signals must be passed between any two nodes in the ring, additional expander boards can be added in parallel to increase the number of lines between units, to a maximum of three expander boards in a single chassis. The 8 line linear switch module is used primarily to provided MIL-STD-1553B switching. The 8 line linear switch board provides eight provided MIL-STD-1553B busses and can connect any or all of the MCP modules in the chassis to any specified bus configuration. If more than eight busses are necessary, additional 8 line linear switch boards can be installed. The 8 line linear switch board contains built-in expansion capability to interconnect multiple chassis. The expansion chassis can connect any of their MIL-STD-1553B MCP modules to the defined busses.

MCPs can be thought of as personality modules used to electrically convert signals into a form compatible with the switching modules. They also provide a connector mount for connectors appropriate to the signal type. Each type of signal being switched must have an MCP. There

are currently the following MCP types: provided MIL-STD-1553B, F16 Weapons Bus, and SMARTNet-2. Anticipated MCP types for future development include: Ethernet, Synchro Input, Synchro Output, 16 Channel Analog Input, 16 Channel Analog Output, Avionics Discrete Input, and Avionics Discrete Output.

Configuration Examples

The flexibility of the MSS modular design provides total freedom to mix and match MCP and switch modules in an MSS. There is no one "right" configuration. The basic approach to configuring the switch system is to analyze the signals to be switched, select the appropriate MCP modules, select the switch modules needed for the MCP types, assign boards to chassis, and connect multiple chassis for large switch arrays. Although there is no MSS requirement to group signals in any particular manner, a large test facility needing multiple units can maximize the number of possible configurations by grouping like signals in same chassis to minimize cable requirements and propagation delays. The following examples show potential configurations.

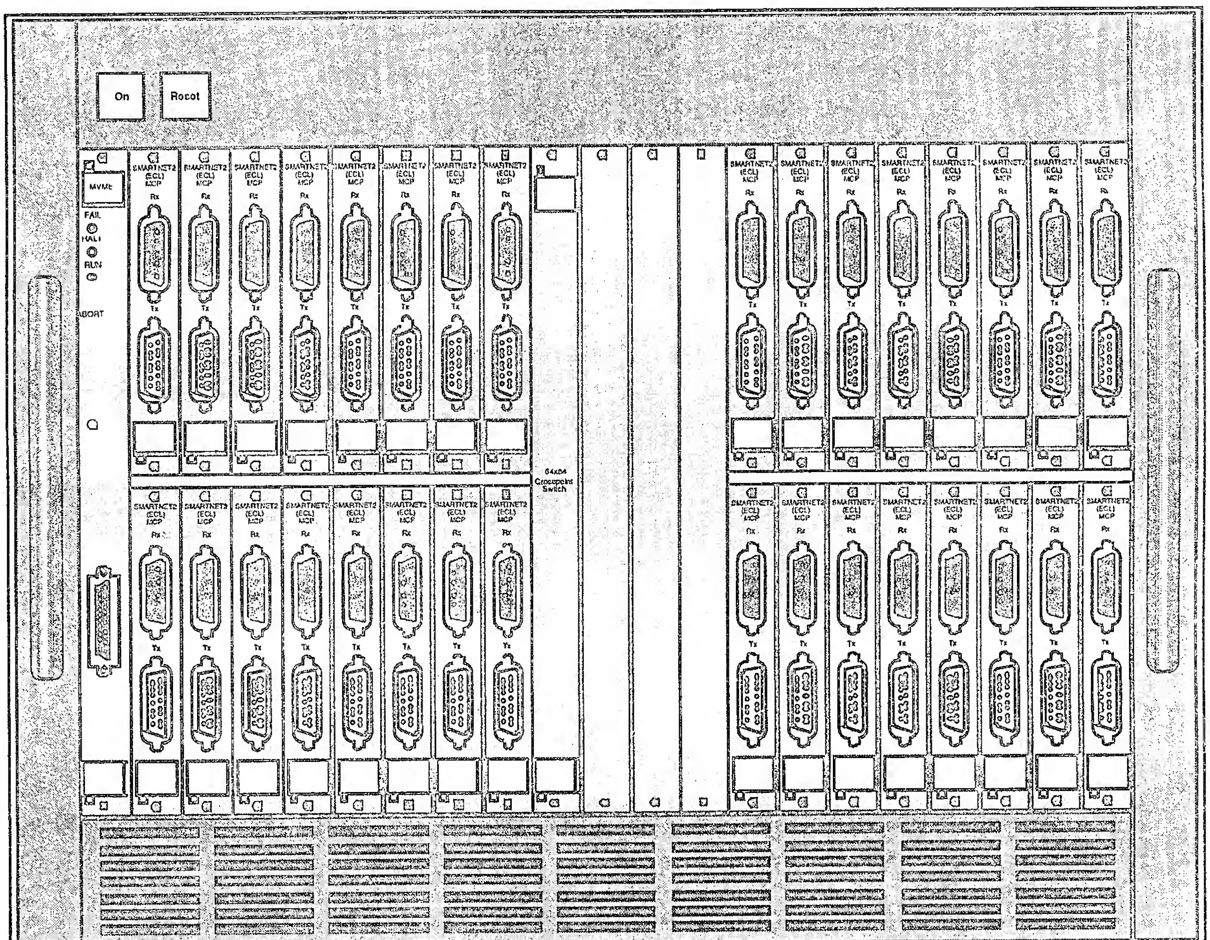


Figure 4. Standalone SMARTNet-2 MSS

Example number one (Figure 4) shows a configuration for a facility with the need to switch a large number of MIL-STD-1553B networks as part of its test requirements. This example is configured to handle at least thirty-two MIL-STD-1553B terminals. The configuration has three 8-line linear switch modules. Each 8-line linear switch can provide eight networks with any combination of MCP connections. Two 8-line linear switches can handle the complete switching needs of a single chassis by providing the capacity to have sixteen networks with two terminals each. The third 8-line linear switch in the configuration is installed on the assumption that at least one more MSS with MIL-STD-1553B MCPs comprises the test facility configuration. The third switch can be connected via the expander ports on the front panel to a remote MSS to provide up to eight MIL-STD-1553B networks which are shared between the units. The third 8-line linear switch can be omitted from the system if no other MSS has MIL-STD-1553B MCPs. Conversely, a forth 8-line linear switch can be installed if more than eight networks must be shared.

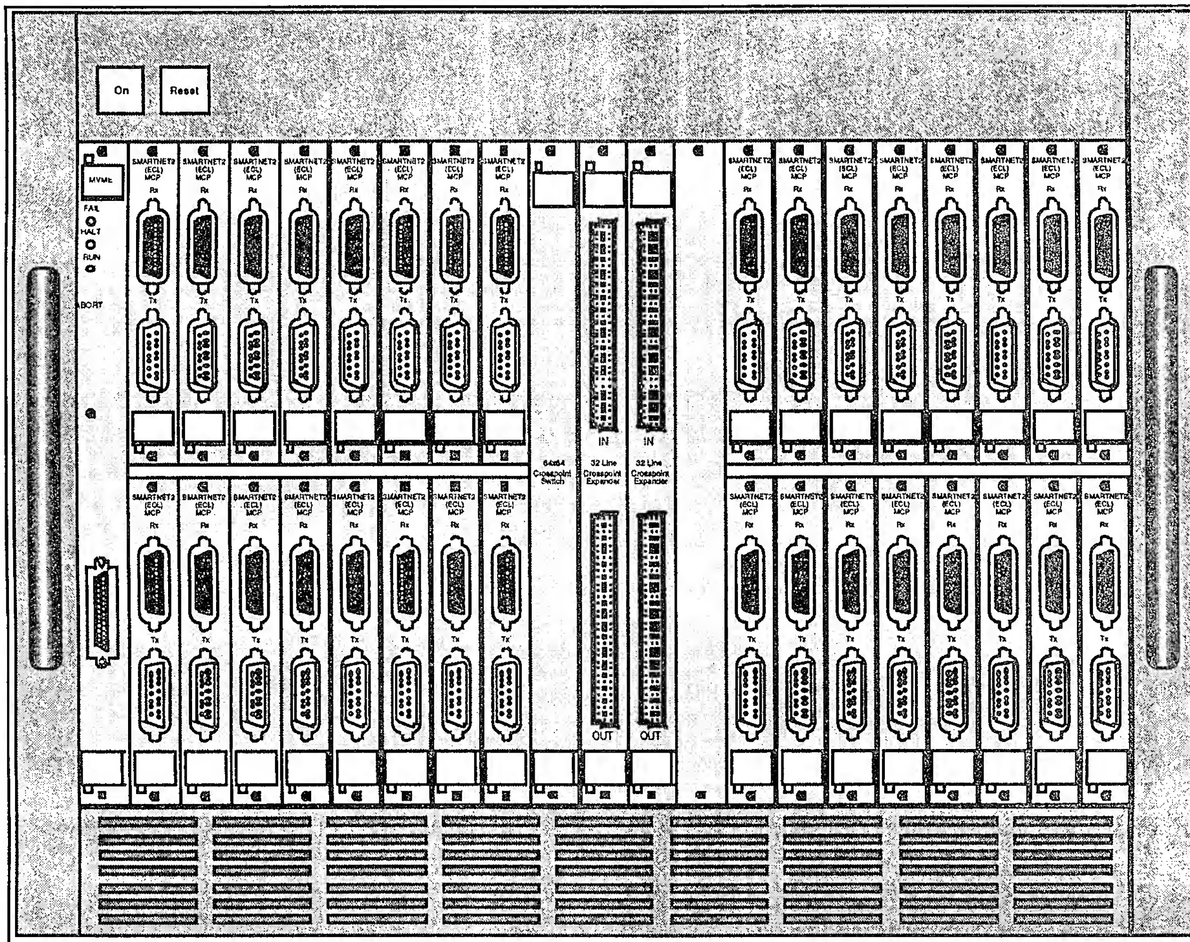


Figure 5. SMARTNet-2 MSS with Expansion Capability

Example number two shows the potential configurations to handle reflective memory networks. Figure 5 shows a configuration which can provide all the possible switching connections to handle up to thirty-two SMARTNet-2 reflective memory interfaces into any number of networks. In this case the only switching component needed is the 64x64 crosspoint switch. Due to the physical size and cost of the switching components in the design, the 64x64 crosspoint switch was not able to include a built-in expansion capability. Expanding the system to incorporate reflective memories beyond thirty-two (Figure 6) simply adds 32 line crosspoint expanders to the MSS with cables between the units. The number of expanders is a function of the number of units and the logic with which the MCPs are assigned to the chassis. If the units are attached to MSSs such that most networks are configured inside a common chassis, then one expander may be all that is necessary. More expanders may be needed if the physical configurations are not optimum. In any event, the firmware will report when a potential configuration exceeds the available expansion capacity. The user has the option of allocating

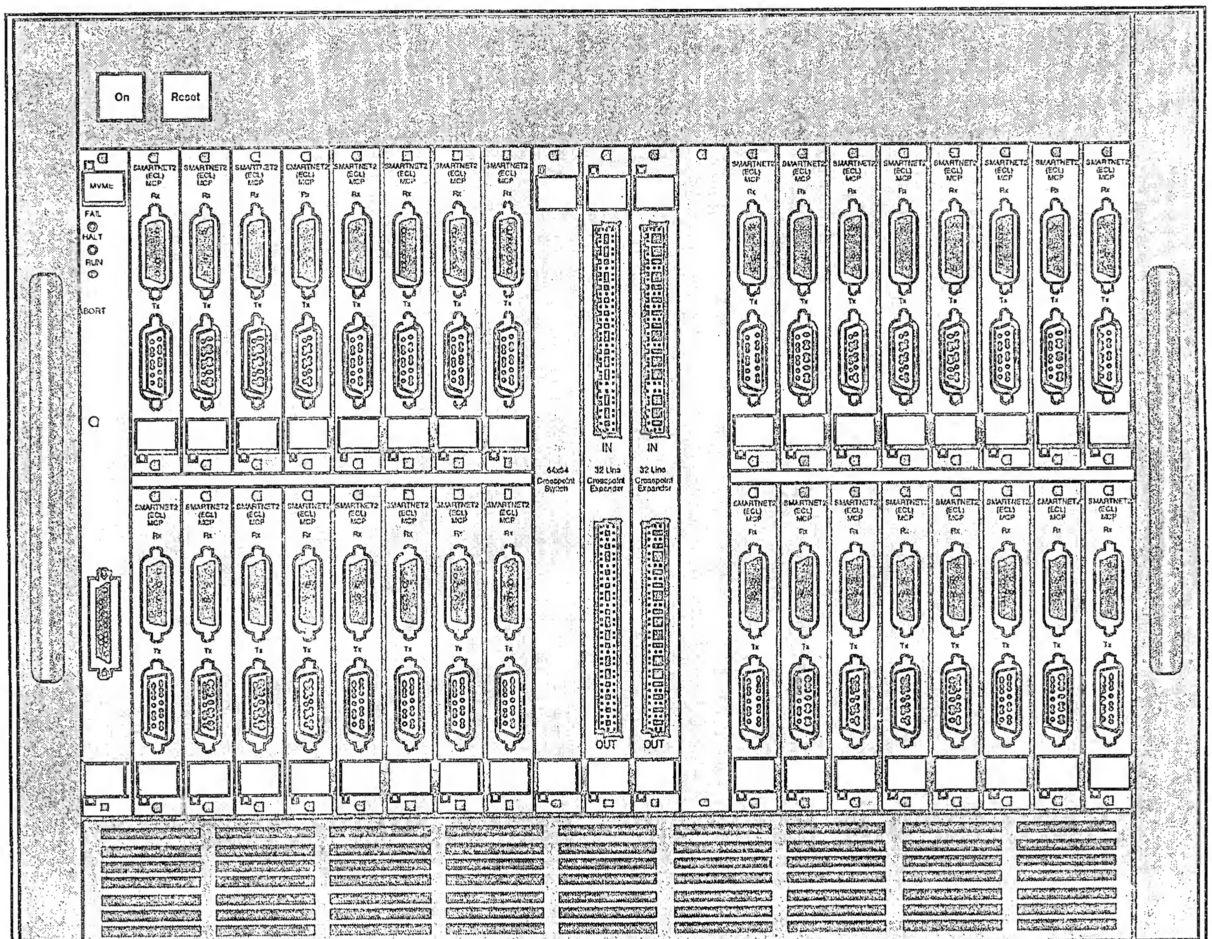


Figure 6. SMARTNet-2 MSS with Expansion Capability

specific expansion lines as he chooses or allowing the optimizing router in firmware to allocate expansion lines for maximum routability.

Summary

The MSS affords the test engineer the flexibility to support efficient management of physical resources as a set of virtual components. This flexibility provides an automated reconfigurable ISF.

The modular design of the MSS is scaleable for the addition of new resources to an ISF. New resource types, varied in format and protocol, can be accommodated with additional MCP module designs. Thus, the MSS modularity provides the expansion capability necessary to support growth in avionics test requirements. As simulation processing requirements grow to improve avionics/environment modeling fidelity, data throughput requirements promise to increase. The wideband design of the MSU is poised to handle this increased demand, as well.

The Digital Architecture Simulator (DAS) Development Effort

Denice S. Jacobs

Avionics Directorate
Wright Laboratory
Wright Patterson Air Force Base, OH 45433-7301
Phone: (513) 255-2766 Fax: (513) 476-4746

Abstract

The ***Digital Architecture Simulator (DAS)*** is an extremely flexible computer architecture simulator which characterizes processor and bus loading effects for any given ***digital*** system architecture. A digital architecture is defined as any system which consists of either homogenous or heterogeneous digital elements (DEs) that execute digital processes (i.e., non-analog operations) in a multiprocessing, multiprogramming environment. The DAS operates as a black box representation of a given digital architecture with respect to time, DAS(t), whereby it accepts predefined DE, network, and process characteristics at time zero, simulates the operation of the architecture as a function of time, and outputs the state of the architecture at user-defined check point(s).

The main objective of this project was to provide an invaluable systems engineering tool which simulated the real-time system performance of any given digital architecture. The need for this unique ***performance trade-off capability*** arose after severe cost overruns and schedule slips were incurred on a major integration effort which involved multiple state-of-the-art Very High Speed Integrated Circuit (VHSIC) signal/data processors and 150,000 lines of real-time software and firmware code. Without this tool, the developer was unable to predict severe operational deficiencies (e.g., system throughput latencies, processor lockouts, and bus contention problems) which unexpectedly surfaced during system integration and test. Hence, it is believed significant payoffs can be obtained when using the DAS during the design phase of any digital architecture development effort, especially when there is no known tool in existence which accomplishes this type of "first look" high level systems analysis with such ease-of-use and design flexibility.

The DAS design consists of four distinct Ada packages which are used to build and execute the subject architecture in a given time period. The different modeling features include:

- Bus contention algorithms such as Least Recently Used (LRU), First Come-First Serve (FCFS), and Static Priority;
- DE process contention algorithms such as Preemptive (interrupt driven), FCFS, and Static Priority;
- Databus transmissions such as packet (serial) and word (parallel) formats;
- Bus direction protocols such as simplex, half-duplex, and full-duplex;
- Input/output (IO) message protocols such as sequential and update rate;
- Bus period specifications (secs/bit);
- Process execution times, null processing times, and context switching (sec);
- Interconnection network topologies using dedicated (static) links and common (shared) buses; and
- An unbounded number of DEs, processes, and buses.

The DAS was specifically written in Ada for ease of code readability and maintainability. In addition, the Ada packages were intentionally designed as separate entities to handle special I/O requirements and different databases which define the hardware characteristics, the process characteristics, and the simulation start condition. Several test cases were executed and carefully scrutinized for accuracy. The simulation results appear to represent actual system performance and highlight expected performance deficiencies such as process starvation, process deadlock, and bus contention problems. Hence, the DAS has successfully met the objectives of this effort.

Introduction

So what exactly is a 'DAS'? Well, in general terms, it is an extremely flexible computer architecture simulator which characterizes processor and bus loading effects for any given ***digital*** system architecture. A digital architecture is defined as any system which consists of either homogenous or heterogeneous DEs which execute digital processes (i.e., non-analog operations) in a multiprocessing, multiprogramming environment. The DAS basically operates as a black box representation of a given architecture with respect to time, DAS(t), whereby it accepts predefined DE, network, and process characteristics at time zero, DAS(t_0); simulates the real-time operation of ***timed DE and Input/Output (IO) events*** for time "n" duration, DAS(t_n); and outputs the "state" of the architecture at user-defined checkpoint(s) which are fixed delta time lengths, Δt_c, such that $\sum \Delta t_c \leq t_n$. A top level block diagram is provided in Figure 1.

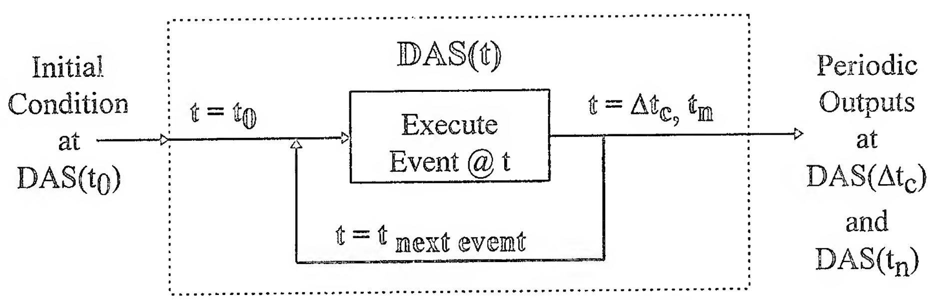


Figure 1. DAS Top Level Block Diagram

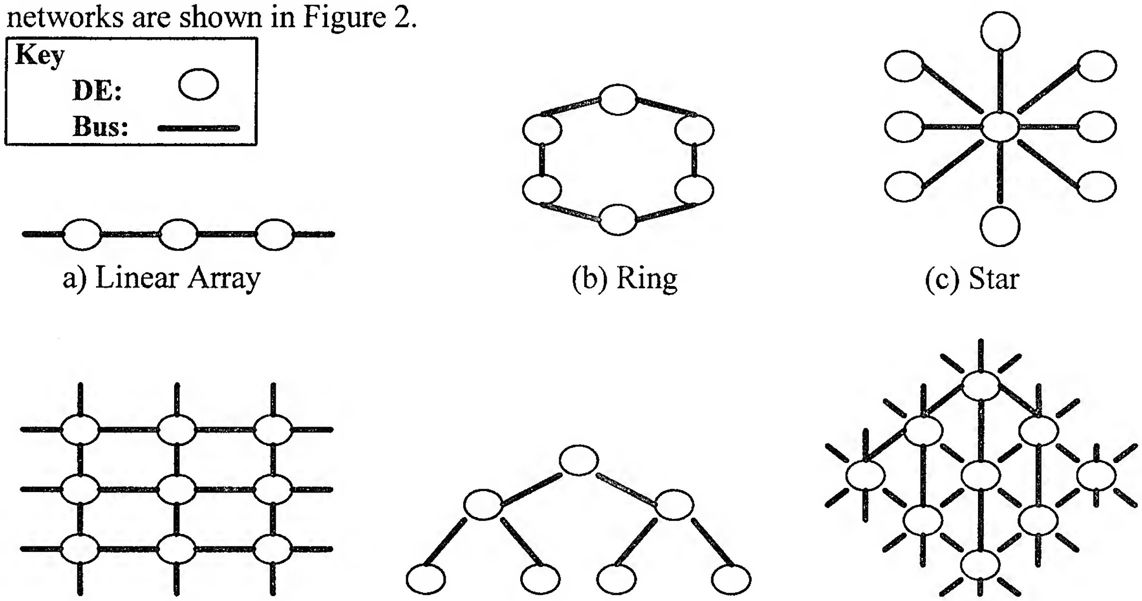
Background

There were three key requirements placed upon the DAS earlier in the design phase of the effort. First, the DAS had to be extremely flexible in terms of its ability to characterize a wide variety of DE and network configurations which accurately model any type of ***multiprocessing environment***. Secondly, the DAS had to implement several different processing protocols and multiple process relationships which realistically simulate any type of ***multiprogramming environment***. And lastly, the DAS had to be written in a language such that it would be ***clearly understood, readily implemented, and easily maintained*** for future enhancements. Consequently, the following architecture/processing features and programming language were thoroughly researched, evaluated, and subsequently implemented in the DAS design.

Networks

The first task was to investigate the different types of network topologies required to model two

or more DEs operating in a multiprocessing environment.[1] There are three basic types of networks, namely, ***static, common, and redundant networks***, which readily meet this requirement. A static network is defined as one which supports "n" bus connections from any DE to "n" different DEs through discrete bus lines (i.e., any given DE can connect to one or more DEs, provided the bus lines do not connect to the same DE twice). Consequently, a static network does not permit any DE to connect to itself or to others more than once. Example static networks are shown in Figure 2.



a) Linear Array (b) Ring (c) Star

d) Near-neighbor Mesh (e) Tree (f) Systolic Array

Figure 2. Static Network Topologies

The second bus configuration type is the common or "shared" network topology. As the name implies, each bus line is shared between three or more DEs such that each DE could control the bus based upon the bus contention algorithm for that particular network. As in the case with the static network, the common network does not permit multiple connections to the same DE. An example of a common network is provided in Figure 3.

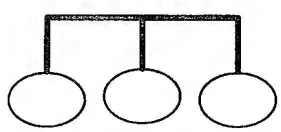
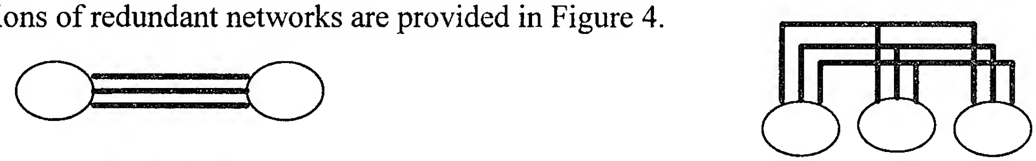


Figure 3. Common Bus Configuration

And finally, the redundant or "backup" network protocol permits two or more bus links between two or more DEs, which applies to both common and static network topologies. Example configurations of redundant networks are provided in Figure 4.



a) Multiple Link Connections b) Multiple Common Bus Connections

Figure 4. Redundant Network Configurations

Data Transfer

The second task was to identify the different rules for data transfer over any given network, namely, ***simplex, half-duplex, and full-duplex*** communications.[2] Simplex communication permits data transfer in only one direction on the bus. Consequently, this type of data transfer could only be implemented on a static or redundant static network where a dedicated bus exists between two DEs. Half-duplex communication permits data transfer in both directions on the bus, but not simultaneously. This protocol could be readily implemented on all three network types. Lastly, full-duplex communication permits simultaneous data transfer in both directions on the bus. Obviously, this protocol would only apply to a static or redundant static network since common bus networks cannot support simultaneous bus operations.

Another area of investigation addressed two types of data transmissions, ***parallel (word) and serial (packet)*** formats, which are traditionally implemented in the subject networks. The parallel message transmission, which consists of one or more data words (i.e., 'x' bits per clock cycle), may equal or exceed the given bus size. However, should the message exceed the bus size, the message is uninterruptable and therefore, the "sending" DE must complete the message transfer. The serial message transmission, on the other hand, consists of one or more data packets (i.e., "y" bits per "y" bus cycles), which equal or exceed the given bus size. The key difference with this format, though, is that higher priority messages can "take over" the network before the original message is complete.

Bus Protocols

The third task was to evaluate the appropriate bus contention algorithms which execute on the subject networks.[3] There were three bus contention algorithms selected: ***First Come-First Serve (FCFS), Static Priority, and Least Recently Used (LRU).*** The FCFS algorithm transfers the first available bus message, regardless of DE priority or recent usage, on any network topology with any type of data transfer protocol. Consequently, in the case where DEs have to share a bus [i.e., (redundant) common networks or half-duplex (redundant) static networks], this algorithm could easily block a slower message-producing DE. The Static Priority algorithm transfers the highest priority bus message, which is assigned the priority of the "source" or sending DE, on any network topology with any type of data transfer protocol. Similar to the previous case, a lower-priority DE could run the risk of being locked out when it competes with higher priority DEs on a shared bus network. The last bus contention algorithm is LRU which permits the least recently used DE to get first access to the shared bus network, and it too can be used on any network topology with any type of data transfer protocol. Of all three protocols, this is the fairest bus contention algorithm; however, it may not be appropriate for every application, especially for higher priority processing requirements.

Process Definition

The fourth task was to define the term "process" as it pertained to a multiprogramming environment.[4] A process is defined as either a ***software, hardware, or firmware entity*** which performs a unique digital function in any given DE for a finite period of time. When a process ends, it may produce up to three types of external stimuli to other processes, including itself, which

represent the following process communication events: 1) an infinite loop; 2) program call; and 3) data notification. In the first case, a process "P1" may signal itself, thereby representing an infinite loop program which runs continually as a background operation. In the second case, two processes "P1" and "P3", may represent a main program which calls to a subprogram, process "P2". And in the last case, process "P1" may notify or signal another process "P2" when data is available. In summary, every process can transmit and receive an unbounded number of stimuli to and from other processes, but only one signal interface may exist between the same process or two different processes. Furthermore, all inputs need to be received before a process can begin executing. Examples of process communications are shown in Figure 5.

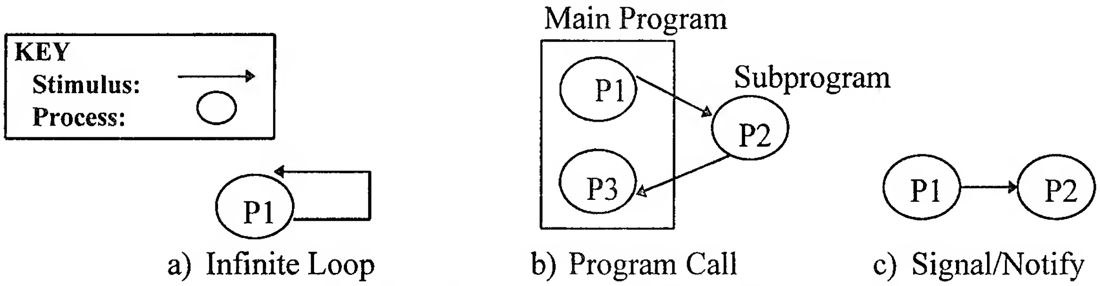


Figure 5. Types of Process Communications

Processing Protocols

The fifth task was to select several processing protocols which accurately modeled a multiprogramming environment, namely, ***FCFS, Static Priority, and Preemptive*** processing protocols.[5] Similar to the FCFS and Static Priority algorithms described earlier in the Bus Protocols Section, a given DE operates on the first available process based upon the FCFS algorithm, and the highest priority process based upon the Static Priority algorithm. Both algorithms are non-intrusive, meaning they do not interrupt the current running process in the given DE. The Preemptive algorithm, on the other hand, can interrupt a current running process if it has a higher priority (i.e., it models an interrupt-driven DE). Consequently, this algorithm, as well as the Static Priority algorithm, can potentially lock out lower priority process(es). The corresponding process state diagram is provided in Figure 6.

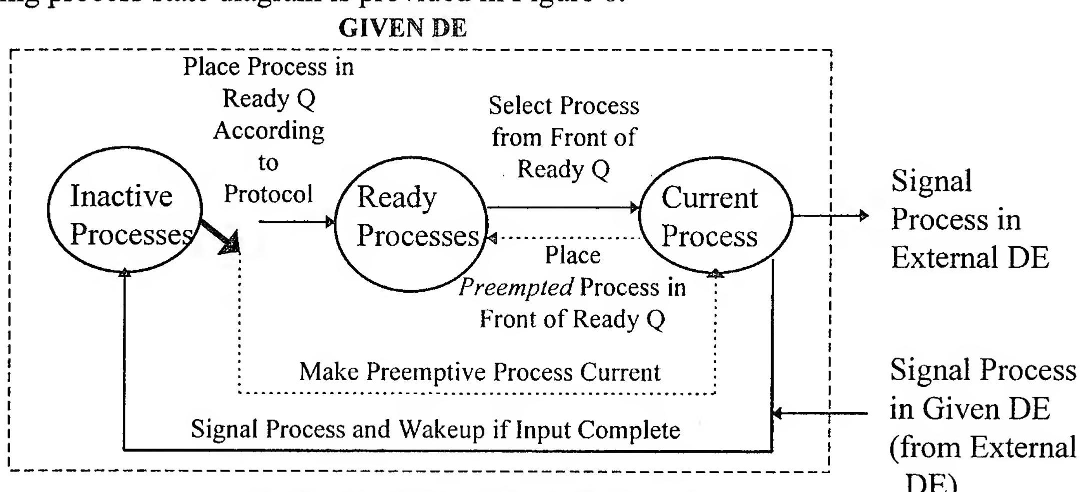


Figure 6. Process State Diagram

IO Queues

The sixth task was to identify two different types of IO queues: The ***sequential and update queue*** algorithms.[6] The sequential queue algorithm simply maintains all of the input "signal" messages in memory which are received from one or more processes, thereby keeping a history of all current and past communication messages. As a result, the sequential algorithm queue models an unbounded circular IO queue which forces the processes to remain "in sync" with one another throughout the passage of time. With respect to the update queue algorithm, it "refreshes" the memory with the most current input messages. Consequently, this algorithm models a fixed queue which provides only the latest information to the receiving processes.

DAS Language

The final task was to select an appropriate programming language which would be clearly understood, readily implemented, and easily maintained for future enhancements.[7] The higher order programming language *Ada* was chosen as it: 1) was specifically designed to be "self commenting"; 2) supported a rigid interface specification and variable/type check protocol; and 3) provided a convenient package concept which permitted the parallel development of the architecture, process, and start condition databases. Hence, Ada was the preferred language of choice as the program was not constrained to execute in real-time in order to simulate real-time operation!

System Description Overview

The DAS consists of four major development areas which were specifically designed to support a wide variety of digital system requirements and performance characteristics. These requirements, namely, the ***Architecture Requirements, Process Requirements, Generic Shell Requirements, and Output Requirements***, are the core foundation from which DAS is able to simulate a realistic multiprocessing, multiprogramming system. Details pertaining to each functional area requirements are provided in the following sections.

Architecture Requirements

The DAS shall model either homogenous or heterogeneous DEs which operate on digital data in a ***multiprocessing environment*** (i.e., two or more DEs). The DEs shall be interconnected with one or more point-to-point links and/or common bus structures which represent the entire bus network of a given architecture. The architecture features shall be defined in a separate database, independent of the process requirements database, such that every feature can be developed without knowledge of the process requirements and their interrelationships.[8] These features include:

- Interconnection network topologies using dedicated (static) links and common (shared) buses;
- Bus direction protocols such as simplex, half-duplex, and full-duplex;
- Bus contention algorithms such as LRU, FCFS, and Static Priority;
- Databus transmissions such as packet (serial) and word (parallel) formats;
- Bus period specification (secs/bit); and
- An unbounded number of DEs and buses, with a minimum of two DEs and one bus structure.

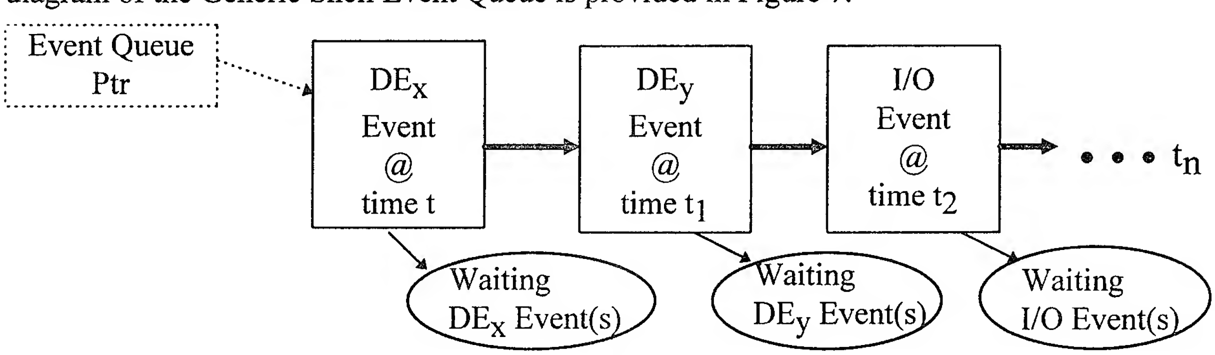
Process Requirements

The DAS shall model digital processes which operate in a ***multiprogramming environment*** (i.e., two or more processes running in the same DE). The DE processes shall communicate with one another via one or more signal stimuli which characterize an infinite loop, program call, or data notification message. The process features shall be defined in a separate database, somewhat independent of the architecture requirements database, such that every process can be defined without knowledge of the network configuration, provided the host DEs are identified in the architecture database. These features include:

- DE process contention algorithms such as Preemptive (interrupt driven), FCFS, and Static Priority;
- IO message protocols such as sequential and update (refresh) rate;
- Identification of the process execution times (sec);
- Identification of the process context switching or scheduling time (sec) per DE;
- Identification of the null process "waiting" time (sec) per DE (i.e., when no other processes are currently running);
- Process communication stimuli to characterize an infinite loop, program call, or data notification; and
- An unbounded number of processes, with a minimum of two processes defined.

Generic Shell Requirements

The Generic Shell is the simulation "heart" or engine of the DAS whose sole purpose is to simulate a real-time multiprocessing, multiprogramming environment based upon the sequential occurrence of ***timed DE and IO events***. A timed event is defined as the "new" state of a given DE or IO message which is placed in a time-ordered Event Queue according to the completion time of the subject event. In other words, the timeline contains the current state of all DEs and any existing IO messages at time "t" which reflect the end times of each event. Consequently, the first event in the Event Queue represents the completion of an operation which is subsequently removed from the Queue and processed (i.e., the next "new" event is selected from the associated waiting events and inserted into the Event Queue), and the "next" first event is removed and processed, and so on, until the simulation ends at time "t_n". A top level block diagram of the Generic Shell Event Queue is provided in Figure 7.



Note: $t \leq t_1 \leq t_2 \leq t_n$

Figure 7. Event Queue Block Diagram

The DAS start condition features shall be defined in a separate database, independent of the architecture requirements database and somewhat independent of the process requirements database, such that every start process exists in the process database but can be defined without knowledge of the architecture structure. These features include:

- Fixed simulation start time at time zero (t_0);
- User-specified checkpoints at delta time lengths (Δt_c);
- User-specified stop time (t_n);
- Simulation of "real-time processing"; and
- Efficient performance (i.e., near real-time execution).

The Generic Shell shall correlate all three databases, namely, the architecture, processes, and start condition databases to create and run the simulation, and shall generate an output data file which contains the checkpoint data and corresponding statistical analysis.

Output Requirements

The DAS shall produce three types of information in the form of an output data file: ***User-specified Checkpoints, Special Event Messages, and Statistical Analysis Report.*** In the first case, User-specified Checkpoints shall be created every delta time point "Δt_c" such that the state of every DE and IO Event in the Event Queue is provided to the user for general information. The DAS shall also provide the capability to review the events at every time increment (i.e., the "end time" of each front event in the Event Queue) whenever Δt_c is set to zero. Special Event Messages, on the other hand, shall be generated as required, to inform the user of key events which may warrant closer inspection through revision of Δt_c. Example "flags" or informational messages include preempted processes, program calls and signals, and completed IO messages. In the last case, a Statistical Analysis Report shall be generated which provides a summary of DE percent utilization, process activation and completion statistics, and IO message notification. In summary, DAS shall provide a comprehensive output file which contains the following detailed information:

- The state of each event in the Event Queue every delta time checkpoint Δt_c;
- The state of each event in the Event Queue at every time increment such that $\Delta t_c = 0.0$;
- A listing of special or key events whenever processes are preempted, a process calls/signals to one or more processes, and/or IO messages are completed; and
- Statistical data in terms of DE percent utilization, process activation and completion statistics, and IO message notification.

System Design Overview

The DAS design consists of four distinct Ada packages which are coordinated by the ***Main Program*** to build and execute the subject architecture for a given time period: ***The Architecture Database Package, The Processes Database Package, The Generic Shell Package, and Unique IO Package.*** The Architecture Database Package reads all of the network characteristics from a user-specified architecture database and places this information into a double linked list or

"node" structure. Similarly, the Processes Database Package reads all of the DE process characteristics and signal relationships from a user-specified process database and places this information into a complex linked list or "process" structure. Next, the Generic Shell Package reads the simulation start condition from a user-specified database, correlates all three subject databases, creates/executes the Event Queue, and generates an output data file. Lastly, the Unique IO Package provides unique IO services to all the subject packages which are not readily supported in Ada (e.g., distinguishing words in a file). A top level interface diagram is provided in Figure 8.

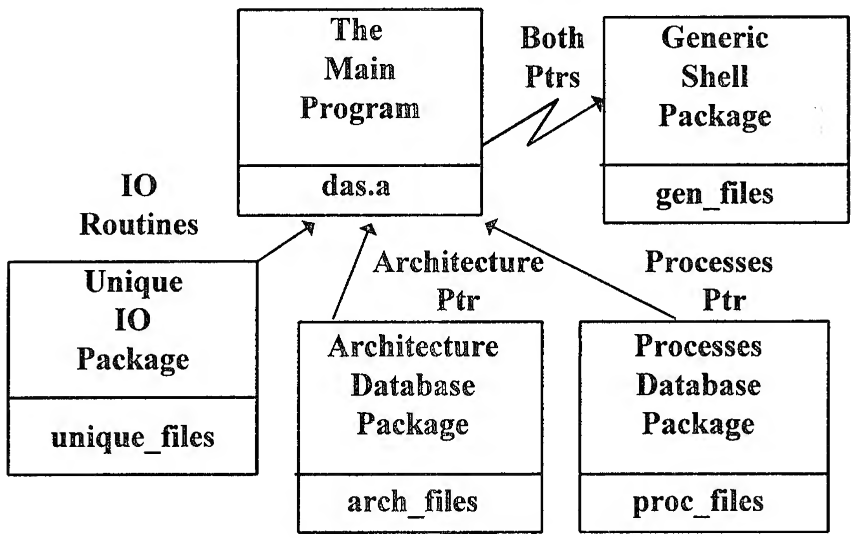


Figure 8. Top Level Interface Diagram

<u>Architecture Database Commands</u>

The Architecture Database input file was specifically designed to be user-friendly in terms of design flexibility, command simplicity, and file documentation. There are two basic input commands, namely, ***LINK: and CBUS: commands***, which retrieve the following information from the subject input file and subsequently load into the node structure: 1) The identification of DEs; 2) Their associated network configuration; and 3) Related bus or path characteristics. As the name implies, the LINK: command is <u>only</u> used to link two DEs together in a static network topology with the associated path characteristics provided in the command line. The path characteristics include:

- Bus Contention Algorithm = (FCFS, Static Priority, LRU);
- Bus Period (secs/bit) = (positive [pos] real number * time unit) per bit, such that time unit = (sec, msec, usec, nsec, psec);
- Databus Transmission = (parallel, serial);
- Databus Size = (pos integer number * bit unit), such that bit unit = (bit, kbit, mbit, gbit); and
- Bus Direction Protocol = (halfdup, fulldup) *(Note: The simplex protocol is implemented by default only if processes communicate in one bus direction between DEs).*

The LINK: command requires the reserved word "LINK:" plus nine data entries which are all separated by at least a single blank space. The command line format is shown in Figure 9, whereby DEx ≠ DEy (*Note: The numbers are provided for sequential clarity;* ***do not*** *enter into database*).

1. LINK:	2. *Bus Contention Algorithm*	3. *Positive Real Number (bus period)*
4. *Time Unit*	5. *Databus Transmission*	6. *Positive Integer Number (databus size)*
7. *Bit Unit*	8. *Bus Direction*	9. *DEx* 10. *DEy*

Figure 9. LINK: Command Format

The CBUS: command is used to connect three or more DEs in a common bus configuration. The command format is ***identical*** to that of the LINK: command with two exceptions: 1) Since the number of DEs is unbounded, a delimiter "%" is required to indicate the end of the command line; and 2) No Bus Direction Protocol is required since a CBUS network operates in a half-duplex mode. The command line format is shown in Figure 10, whereby DE1 ≠ DE2 ≠... DEn.

1. CBUS:	2. *Bus Contention Algorithm*	3. *Pos Real Num (bus period)*
4. *Time Unit*	5. *Databus Transmission*	6. *Pos Int Num (databus size)*
7. *Bit Unit*	8. *DE1 DE2 ∘∘∘ DEn %*	

Figure 10. CBUS: Command Format

Both command line formats are case insensitive (i.e., all letters are either uppercase or lowercase) and permit one-line comments in the database which are uniquely identified with a star character "*" followed by at least one blank. The user may <u>begin</u> with either command; however, each <u>successive</u> command must ***build*** upon previous ones, such that at least one DE listed in the current command line is defined in an earlier command (i.e., the commands are not mutually exclusive). And lastly, the Architecture Database Package creates an ***Architecture Pointer*** that specifically points to the "beginning" of the node structure (i.e., the bus network defined by the first command in the file) which is then passed onto the Main Program for further processing.

<u>Processes Database Commands</u>

The Processes Database input file was also designed to be easy-to-use and extremely flexible in terms of system characterization and file documentation. There are two basic input commands, namely, ***PROCESS and MESSAGE: commands***, which retrieve the following information from the subject input file and subsequently load into the process structure: 1) Identification of the host DEs; 2) Their associated processing requirements; and 3) The IO message relationship between processes. In particular, the PROCESS: command is used to specify the following DE process characteristics which are directly loaded into the process structure:

- Host DE Name;
- Process Contention Algorithm = (FCFS, Static Priority, Preemptive);
- IO Message Protocol = (update, sequential);
- Null Processing Time = (positive real number * time unit);
- Context Switching Time = (positive real number * time unit);
- Process Priority = positive integer; whereby 1 = highest priority; and
- Process Execution Time = (positive real number * time unit).

The PROCESS: command requires the reserved word "PROCESS:" plus several data entries which are all separated by at least a single blank. A delimiter "%" is required at the end of the subject command line since the number of processes must be unbounded. It should also be noted

that the process priority is a required input for each process, regardless of the chosen contention algorithm. The command line format is shown in Figure 11, whereby P1 ≠ P2 ≠ ... Pn.

1. **PROCESS:**		2. *Host DE Name*	3. *Contention Algor*	4. *IO Msg Protocol*
5. *Pos Real Num (null)*		6. *Time Unit*	7. *Pos Real Num (context)*	8. *Time Unit*
9. *P1*	*Priority*	*Pos Real Num (execution)*	*Time Unit*	
P2	*Priority*	*Pos Real Num (execution)*	*Time Unit*	
:	:	:	:	
Pn	*Priority*	*Pos Real Num (execution)*	*Time Unit*	%

Figure 11. PROCESS: Command Format

The purpose of the MESSAGE: command is to identify the signal interface between the source process and one or more destination processes. The total message size sent between the source and any given destination process is described in terms of a positive integer number and bit unit (i.e., bit, kbit, mbit, and gbit), and is loaded into the IO Message Record. The "relationship" between the source and destination processes (i.e., infinite loop, program call, or data notification) is simply created by listing the source process first, followed by the initial destination process and corresponding message size, followed by the next destination process and corresponding message size, and so on, until all of the relationships stemming from the source process are complete. A delimiter "%" is required at the end of the command line since the number of process calls are unbounded. The command line format is shown in Figure 12, whereby P2 ≠ P3 ≠ ... Pn.

1. **MESSAGE**:	2. *P1 (source process)*		
3. *P2 (first destination process)*	*Positive Integer Number (IO msg)*	*Bit Unit*	
P3 (second destination process)	*Positive Integer Number (IO msg)*	*Bit Unit*	
:	:	:	
Pn (n-1 destination process)	*Positive Integer Number (IO msg)*	*Bit Unit*	%

Figure 12. MESSAGE: Command Format

Both command line formats are case insensitive and permit one-line comments in the database which are uniquely identified with a star character "*" followed by at least one blank. The user may begin with either command; however, after the database is complete, all the processes listed in any given MESSAGE: command must be identified in one or more corresponding PROCESS: commands but not (necessarily) visa versa. Furthermore, the PROCESS: and MESSAGE: commands must uniquely identify a process relation and host DE characteristics, respectively, such that they cannot be enhanced or "added to" by subsequent commands. This design feature was specifically implemented to prevent the user from creating enhanced commands which could be added haphazardly throughout the file and subsequently overlooked, especially if the database were very large. And lastly, the Processes Architecture Database creates a ***Processes Pointer*** that points to the beginning of the process structure (i.e., the process record defined by the first command in the file) which is then passed onto the Main Program for further processing.

Generic Shell Command

The Generic Shell Package has a fairly simple input command file format which requires a single

command word called the *START: command*. This command basically requests a delta check time, a simulation end time (i.e., both in terms of a positive real number * time unit), and the name of the process(es) to begin at time zero. A delimiter "%" is also required at the end of the command line since the number of start processes are unbounded. The command line format is shown in Figure 13, whereby P1 ≠ P2 ≠ ... Pn.

1. START:	2. *Positive Real Number (delta time)*	3. *Time Unit*
	4. *Positive Real Number (end time)*	5. *Time Unit*
	6. *P1 P2* ooo *Pn %*	

Figure 13. START: Command Format

The subject command line format is case insensitive and permits one-line comments in the database which are uniquely identified with a star character "*" followed by at least one blank. The user may only use one START: command; new information or "enhancements" may not be added with additional START: commands. Furthermore, at least one start process must be identified in the command and none of start processes may reside in the same DE. The user-defined delta checkpoint may be defined as 0.0 secs, such that the status of every event record in the Event Queue is provided to the user, but it may not be greater than the simulation end time.

Conclusion

The DAS was developed to accurately characterize a real-time multiprocessing, multiprogramming environment for a wide variety of digital applications. The system design was based upon realistic processing characteristics and architecture features such as unique network topologies, multiple process and bus contention algorithms, and different IO transmission formats. After several test cases were executed and carefully examined for accuracy, the simulation appears to represent actual system performance and highlights expected performance deficiencies.

References

1. Tanenbaum, A. S., *Computer Networks*, Prentice-Hall, Inc., New Jersey, 1981.
2. Niedermiller-Chaffins, D., and Heywood, D., *Networking Technologies*, New Riders Publishing, Indiana, 1993.
3. Hwang, K., and Briggs, F.A., *Computer Architecture and Parallel Processing*, McGraw-Hill, Inc., New York, 1984.
4. Gehani, N., and McGettrick, A. D., *Concurrent Programming*, Addison-Wesley Publishing Company, England, 1988.
5. Peterson, J. L., and Silberschatz, A., *Operating System Concepts*, Second Edition, Addison-Wesley Publishing Company, Massachusetts, 1986.
6. Hamacher, V. C., and Vranesic, Z. G., and Zaky, S. G., *Computer Organization*, McGraw-Hill Book Company, New York, 1978.
7. Sebesta, R. W., *Concepts of Programming Languages*, The Benjamin/Cummins Publishing Company, California, 1989.
8. Vasta, J. A., *Understanding Database Management Systems*, Second Edition, Wadsworth publishing Company, California, 1989.

ALLOCATION OF SOFTWARE IN A DISTRIBUTED COMPUTING ENVIRONMENT

Dan H. McMillan
Jeffrey A. Van Fleet
Science Applications International Corporation
Dayton, Ohio 45432

Abstract

The Avionics Logistics Branch at Wright Laboratories (WL/AAAF) and SAIC are currently developing a new type of distributed processing architecture for high performance real-time applications called the Virtual Test Station (VTS). The VTS architecture consists of a collection of computers, workstations, and dedicated application hardware that can be dynamically reconfigured to meet changing and/or multiple simultaneous real-time processing applications. When a VTS user needs to run an application, he selects the desired software for that application and the VTS Resource Allocation Manager (RAM) automatically identifies the resources needed for that application, interconnects them, and downloads the selected software. The resource allocation process optimizes the selection of the resources and partitions the software across those resources. When the application session is complete, the RAM removes the downloaded software, breaks the inter-resource communication links, and makes the resources available for new applications. This paper describes the approach we have implemented for resource selection and software allocation in the VTS.

VTS Overview

To keep pace with the growth of avionics software support costs, particularly for large, complex avionics systems, WL/AAAF and SAIC are building on the previous Advanced Multi-Purpose Support Environment (AMPSE) work with the development of the Virtual Test Station. The AMPSE is a distributed architecture containing a set of heterogeneous computers connected via a reflective shared memory network, either the Shared Memory Architecture Real-Time Network (SMARTNet) or SCRAMNET. The AMPSE architecture was designed for testing avionics, therefore the architecture supports the avionics-under-test with the remaining aircraft simulated using software models, software emulators, and hardware emulators. While the AMPSE approach has made the development of test stations faster and less expensive, the VTS takes avionics support environments to the next level. In the AMPSE, all of the avionics and processing computers are hard-wired together with SMARTNet and MIL-STD-1553. As shown in Figure 1, the VTS takes these same signals into an software controlled electronic switching device, the Modular Switching System (MSS). This allows the VTS to provide an OFP test engineer with a complete set of avionics and processing computers that are needed for any given test, and it frees the remaining hardware for use by other OFP engineers. A major cost driver of avionics support environments is the expense and the underutilization of the avionics and

processing computers used in these systems. One of the primary goals of the VTS is to best utilize the hardware and software to provide each OFP engineer with a "virtual" test station. (i.e., the subset of resources and software necessary for their avionics test). This is the role of the Allocator. It uses the pool of available avionics and processing computers, the collection of simulation software, the simulation base frame rate, and the selected avionics-under-test to determine an optimum set of resources for the OFP engineer's test. This paper focuses on the Allocator algorithm that generates this optimum set of resources.

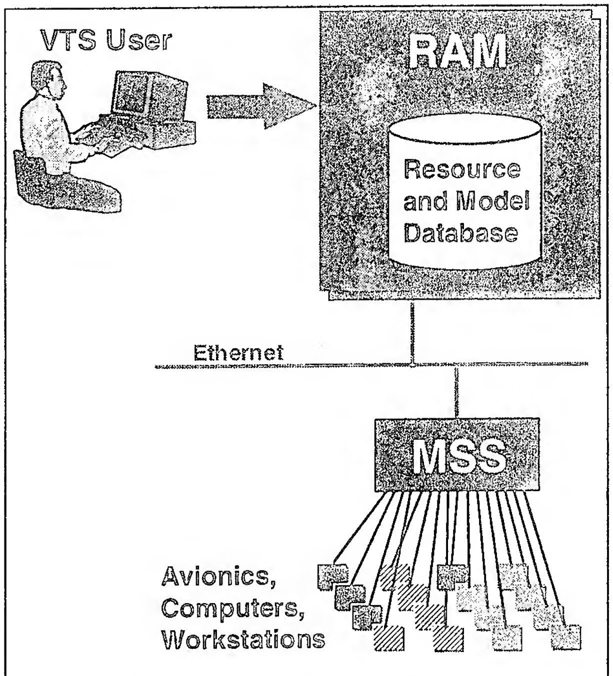


Figure 1 VTS Block Diagram

Design Considerations

Using the Resource Allocation Manager (RAM), an OFP Engineer can simply select an avionics device for testing and the simulation models necessary for the test. The engineer doesn't have to specify which computers are needed or which software models run on each computer. The Allocator utilizes a database of this information that identifies every hardware resource (e.g., computer, workstation, and avionics device) known to the VTS, and when each is reserved for another test session. The database also identifies every software model that is part of an avionics simulation. Since the basic VTS architecture is a distributed simulation (i.e., software models running on separate, yet synchronized computers), this database contains a cross-reference between software and hardware platforms to identify which computer can run each model. After the OFP engineer selects the avionics device and simulation models, the Allocator uses the database and generates the smallest set of resources needed to execute the selected models.

There are some basic issues to consider when attempting to perform this function. For us, the primary consideration was *finding a solution if one exists.* We investigated numerous approaches, and settled on a single pass solution that sorts the models to optimize the chance of finding a solution on the first pass. *Performance* was our second consideration. To enhance performance, we preprocess all static data to accelerate the decision making processes. We use fast response data structures like array and array slice arithmetic instead of searching through linked lists, and we minimize disk access by loading all data prior to Allocation. Our last design consideration was to *find the optimum solution.* Determining what optimum meant was a critical step. For us, it meant finding the solution using the minimum number of resources, while maximizing their availability (i.e., available for the greatest length of time). Finding a solution using the minimum number of resources is important because it makes more resources available for other OFP Engineers to perform simultaneous test sessions. Maximizing the resources'

availability gives the OFP Engineer the most flexibility when scheduling a the test session. The next few paragraphs describe the algorithm that we implemented in meeting these objectives.

Design Overview

As shown in Figure 2, the Allocator receives the hardware resources, the user's software selections, and the schedule data as inputs. The schedule data is composed of the session start time, the session duration (i.e., how long the session is to continue), and the availability of all resources.

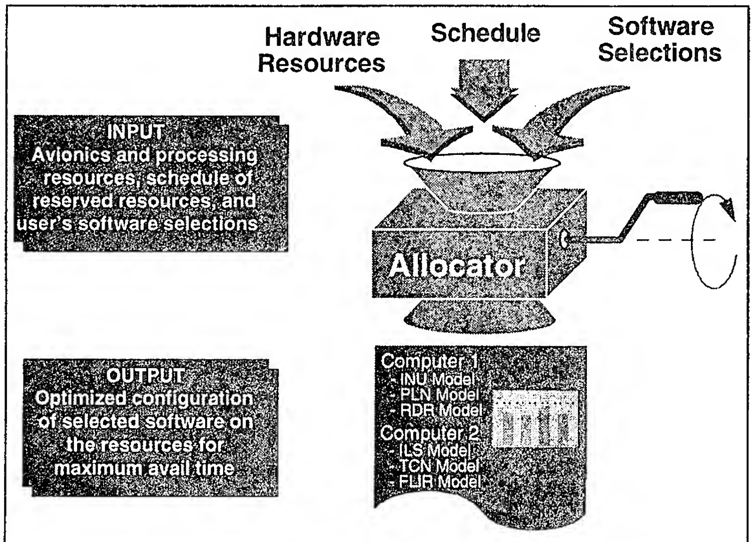


Figure 2. Allocator Overview

The Allocator takes these inputs and computes an optimized configuration and a maximum session duration. The configuration is composed of the avionics devices, computers (i.e., processing resources), and simulation models that are assigned to each computer to satisfy the OFP Engineer's selections. After a configuration is generated, the resources are reserved for the scheduled time period which guarantees they are not used by other OFP Engineers. If the algorithm cannot generate a configuration, the user will be notified.

Allocator Algorithm

The Allocator algorithm is a single pass algorithm that calculates resource availability, sorts the simulation models, chooses computers, and distributes the models among those computers. The next several paragraphs describe these functions.

Calculating Resource Availability - Since the RAM gives OFP engineers the ability to schedule test sessions in the future, we must track the reservations for each avionics and processing resource (i.e., computer). Thus, when a user selects a specific avionics box, the RAM knows immediately if another user has it reserved. As discussed earlier, one of the main functions of the Allocator is to maximize the availability of the whole configuration (i.e., the period of time that the allocated resources are all available). When the Allocator begins, it checks each resource to make sure it is online (i.e., not down for maintenance) and it is not reserved at the selected start time. Then, it calculates its availability and adds it to the available resource table. Availability is the difference between the start time and its next chronologically reserved time (see Table 1 for an example of an Allocator generated available resource table). If the resource is offline, or the session's start time conflicts with a reserved time, its availability is zero.

Sorting the Models - The next step the Allocator performs is to sort the models in such a manner that it optimizes the solution and gets a best fit on the initial try. The algorithm sorts the models so that those that only run on a single resource are first, and those that run on several resources are last. The theory is that those models with comparatively limited distribution possibilities need to be distributed first. We would not want to fill up critical resources with models that can run on a large variety of resources.

Table 1. Generated Resource Availability

Test Session start time scheduled for 06-13-95 at 0700

Res ID	CPUs	Reserved Times (date,hr-hr)	Avail (hrs)
1	030, 040, SPARC	06-13-95,1500-1900	8
2	VAX3100	06-13-95,2300-2400	16
3	SPARC	06-13-95,1900-2300	12

Choosing Computers and Distributing Models - The final step the Allocator performs is to choose the computers (i.e., processing resources) and distribute the models on them. Here, the Allocator uses the database to determine the set of computers that can execute each model. It takes this list and eliminates those computers which are not available, then eliminates those already in the configuration (if this is the first resource, none are in the configuration). If all have been eliminated, then the Allocator reports back to the OFP engineer that the resources required to satisfy the user's selections are not available.

Using the remaining resources, the algorithm determines the next resource to add to the configuration. To do this, it first finds a resource whose availability equals the current configuration's availability. If none exist, then it finds the resource that is greater than the current configuration's availability, but is the closest. If none exist, it finds the resource that is the closest to the current configuration's availability, and the current configuration's availability is reduced accordingly. This process maximizes the availability of the whole configuration.

It adds this computer to the configuration, and assigns the model to that computer. The Allocator then steps to the next model in the sorted model list. The algorithm attempts to assign this model to a computer already in the configuration, provided the total of all the models' execution times assigned to that computer does not exceed the simulation frame time. If it exceeds the frame time, it is prevented from being placed on this computer, and the Allocator attempts to put the model on another computer already in the configuration. If it cannot, it uses the above technique to choose another computer. This process continues until all models have been assigned to computers.

Additional Features

To provide additional flexibility for the OFP Engineer, the basic Allocator algorithm has been updated to accommodate multi-CPU processing computers (e.g., a VME chassis with several

single-board computers). Additional features were also added to help the engineers responsible for maintaining the VTS software and hardware. To support integrating and testing new VTS hardware resources, a system maintainer can select specific processing computers and manually distribute software to the CPUs. To support VTS software development and maintenance, the system maintainer has the additional capability of selecting a class of computer and placing software on it. For example, this feature would be used if the system maintainer wants to test a new simulation model that runs on a VAX under the VMS operating system. Several VAX-VMS computers may exist in the VTS, so instead of specifying which one he wants, he can select the computer class "VAX-VMS" and the Allocator will find one with the best availability. This reduces resource conflict, thus increasing the VTS flexibility. This new Allocator algorithm is still a single pass algorithm, sequentially performing the same operations as before. However, in addition to these operations, a significant amount of the database is now preprocessed to meet our performance requirements. The following paragraphs describe the functionality that has been added to the Allocator to support the new features.

Preprocessing - Because performance is critical, the algorithm's static input data is preprocessed prior to invoking the Allocator. The database of VTS information has been expanded for the simulation models to include the model's execution time for each type of CPU, and a series of four fields for improving the Allocator's performance. Because the VTS architecture is based upon a series of distributed hardware resources, each model may be executed on several computers (i.e., processing resources). The first three fields provide sums of all the computers, CPUs, and execution times for each model. The final field is a list of the specific computers that the model can run on. Refer to Table 2 for an example of this data.

Table 2. Software Model Data

Model	68030	68040	VAX3100	SPARC	# Res	# CPUs	Exec Time	Sorted Resources (Res ID>CPUs)
ADC	5000	1000	5000	0	2	3	11000	1>040,030 2>VAX3100
ENV	7000	4000	8000	6000	3	5	31000	1>040,SPARC,030 2>VAX3100 3>SPARC
FCN	1250	800	0	0	1	2	2050	1>040,030
FCR	10000	8000	0	0	1	2	18000	1>040,030
HUD	8000	3000	0	1500	2	3	14000	1>SPARC,040,030 3>SPARC
INU	3500	2570	3000	1000	3	5	11070	1>SPARC,040,030 2>VAX3100 3>SPARC
PLN	7000	4900	0	0	1	2	11900	1>040,030
SMS	9000	7000	0	0	1	2	16000	1>040,030
WEP	0	0	0	4000	2	2	8000	1>SPARC 3>SPARC

Calculating Resource Availability - No changes were necessary to accommodate the new features. This function returns a table of resource availability, as shown in Table 1.

Additions to Sorting Models - The goal of model sorting is still to optimize the configuration and get the best fit on the first try. It still sorts the models such that the most limited are first. For example, a model that only runs on one VTS computer is at the top of the sort. Previously, this algorithm only sorted the models based on the computers they can run on. Now, the algorithm is modified to account for the preprocessed CPU sum and the execution time sum. The Allocator reduces these sums for resources that have an availability of zero. The sums indicate the likelihood of finding a resource that can execute a model, and the algorithm sorts these models based upon that likelihood. By reducing the sums for non-available resources, the model sort is more accurate. The primary sort criteria is the number of processing resources on which a model can execute. It is sorted from those that execute on the least number of resources to those that execute on the greatest number of resources. If necessary, the second sort criteria is the number of CPUs that the model can run on, again from least to greatest. The final sort criteria (if needed) is based on execution time (i.e., the sum of the execution times across all CPUs that this model can execute on). It is sorted from largest to smallest. This last criteria is based on the theory that larger sums imply that the execution time per CPU is higher. Therefore, it occupies more of the CPU's processing power and makes it more difficult to assign. Using the Model data in Table 2 and the Resource Availability in Table 1 as input, the following is the sorted model list that is generated using this algorithm:

FCR, SMS, PLN, FCN, WEP, HUD, ADC, ENV, INU

Additions to Choosing Computers and Distributing Models - In the basic Allocator (described earlier), the engineer had the capability of only selecting an avionics device and simulation models. The Allocator built the configuration starting with the avionics devices, then added the processing resources that executed the selected models. Now, the Allocator must account for the user selecting CPU classes and specific computers. This results in the following sequence of events:

1) Assign the selected avionics to the configuration. Since the user has explicitly specified these avionics, they must be part of the configuration. If any of these are unavailable, then a configuration cannot be generated, and the Allocator reports this to the user.

2) Assign selected processing resources (and their CPUs) to the configuration. If the user has indicated that models are to run on specific CPUs within these resources, the models are also assigned to these CPUs. Again, if any are unavailable, then a configuration cannot be generated, and the Allocator reports back to the user.

3) Allocate resources for user selected CPU classes (the user may have assigned models to these). This is a complex step because the Allocator must find a CPU of the correct class and it must have enough room to assign the models that the user requested. To most efficiently utilize the VTS resources, the Allocator will first try to find a CPU of the requested class in the existing configuration. If found and the models fit, then the Allocator simply adds those models to that CPU. If not found or the models do not fit, then an entire resource (one containing the

requested CPU class) is chosen and added to the configuration and the models are assigned to that CPU.

4) Distribute the models that were not assigned to a specific processor. This step is the same as described in the basic Allocator algorithm. First, the algorithm will attempt to place the model on a CPU in the existing configuration. If the model can't be placed, then a new resource will be chosen.

Use of the Allocator algorithm yields a configuration that contains a highly optimized set of resources, their CPUs, and the simulation models distributed across the CPUs. Table 3 illustrates the configuration that was generated using the Resource Availability from Table 1 and the Model database in Table 2 (this assumes all of the models were selected by the user).

Table 3. Generated Configuration

Generated Configuration (for a frame time of 20,000 microseconds)									
Res ID	CPU	Model	Exec Time	CPU	Model	Exec Time	CPU	Mode 1	Exec Time
1	040	FCR	8000	030	FCN	1250	SPARC	WEP	4000
		SMS	7000		HUD	8000		ENV	7000
		PLN	4900		ADC	5000			
					INU	3500			
Total			19900			17750			11000

Availability: 8 hours

Future Enhancements

Since development of the Allocator algorithm, several enhancements have been identified. They include updating preprocessed sums when a resource is added to the configuration, shuffling other users' reserved resources, and allowing lower rate models to start on different frames.

Update Sums - After a resource is allocated to the configuration, the algorithm would update the sums and resort the remaining unassigned models. The theory behind this is one less resource is available to the models, thereby changing the order in which the remaining models should be allocated.

Shuffling other users' resources - If a critical resource was reserved for another configuration, the Allocator would unreserve that resource if it could generate a new configuration for that user. The resource would then be allocated to the current configuration. This helps eliminate some potential conflicts and promotes simultaneous test sessions.

Different rate models - The algorithm currently does not take into account different hertz rate models. It assumes a peak usage of CPU for each frame. The algorithm could be modified to put different rate models onto the same CPU, controlling their starting frame. This would keep the peak load for each frame beneath the frame time limit, and potentially utilize less resources.

Conclusion

The VTS concept represents a very powerful tool for configuring, scheduling, and managing distributed architectures in all levels of avionics software testing. The Allocator algorithm is the backbone of the VTS architecture. It generates an optimized configuration of avionics and processing resources to meet the OFP Engineer's needs. It considers the engineer's selections, simulation base frame rate, execution times of software across multiple platforms, and availability of resources when determining the loading of software across its processing resources. It greatly enhances the VTS flexibility by defining the minimal set of resources for any given simulation, thus providing a better opportunity for simultaneous test sessions. It is a robust and flexible algorithm that has applicability to any environment where software needs loaded across a distributed set of heterogeneous computing resources.

VTS (VIRTUAL TEST STATION) TECHNOLOGIES

Steven A. Walters
Science Applications International Corporation
Dayton, Ohio 45432

Abstract

The Avionics Logistics Branch at Wright Laboratories (WL/AAAF) and SAIC have been conducting research to reduce the cost and improve the performance of simulators used to integrate and test avionics software. This research has led to the development of the VTS (Virtual Test Station). The VTS automates the configuration of avionics integration support facility resources to meet the changing needs of avionics engineers and testers as their avionics software progresses from static unit testing to full dynamic integration testing. The two central elements of the VTS architecture are the MSS (Modular Switching System) and the RAM (Resource Allocation Manager). In the VTS, the communication signals from a set of modular, distributed test station resources are routed through the MSS permitting resources to be interconnected as independent groups under remote control. The RAM provides an intuitive, graphical interface to users and controls the automatic selection, scheduling, and allocation of the test station resources. The VTS permits multiple groups of engineers to share a common set of testing resources, thereby promoting more efficient resource utilization with a smaller aggregate set of resources. This reduces both the acquisition and maintenance costs of the avionics support facility, particularly for large complex avionics systems, while providing avionics developers and test engineers with the exact configuration they need at any time during the avionics development and testing cycle. This paper describes the technologies we have employed in creating the VTS.

Introduction

The VTS Concept is designed for a large avionics integration support facility (ISF) or system integration laboratory (SIL) that requires a number of avionics test stations for supporting a large weapon system with many embedded computers. VTS is also intended for a facility that must support a number of embedded computers across a variety of different weapon systems. Taken together, there is a significant quantity of resources, such as simulation computers, workstations, line replaceable units (LRUs), etc., in a large ISF or SIL. Unfortunately, in a typical ISF or SIL there is little or no cross utilization or sharing of these resources across the different test stations. This results in a great deal of duplication from test station to test station and also a loss of efficiency when the work load on a particular test station approaches 100%.

The VTS Concept addresses this problem with two basic functions. First, we route all of the communication signals from the test station resources to a switching device so that groups of

resources may be interconnected under remote control. Second, we provide a straightforward, intuitive way to access these resources and to combine arbitrary groups of them to form "virtual" test stations. "Virtual" in this context does not imply a relationship to the now popular "virtual reality". It means that the particular physical resources that provide a test station function are not fixed, but may vary from one test session to another. The test station functionality will be preserved from one test session to another, but it may be mapped to different physical resources during different sessions due to variations in availability based on other users, maintenance, equipment failures, etc. The result is that with VTS we make more efficient use of our test station resources. This in turn reduces the total number of resources needed to support an avionics suite which lowers the acquisition and upgrade costs for the facility and decreases facility maintenance costs. Most importantly, however, we reduce both the time and the cost of testing operational flight program (OFP) software.

We have implemented the two basic functions of the VTS Concept, switching and resource management, with two major components, the Modular Switching System, or MSS, and the Resource Allocation Manager, or RAM, respectively (Figure 1). The MSS is a hardware device with switching modules and communication interface modules for electronically connecting inter-resource communication signals, such as high-speed fiber-optic and twisted-pair networks (SCRAMNet, SMARTNet, etc.), low-speed networks and busses (Ethernet, MIL-STD-1553, etc.), and analog/discrete/synchro signals. The connection of resources within the MSS is directed by the RAM through messages delivered via Ethernet. The RAM is software that resides on modern, UNIX-based workstations and provides users with an intuitive, graphical toolset for selecting and scheduling resources.

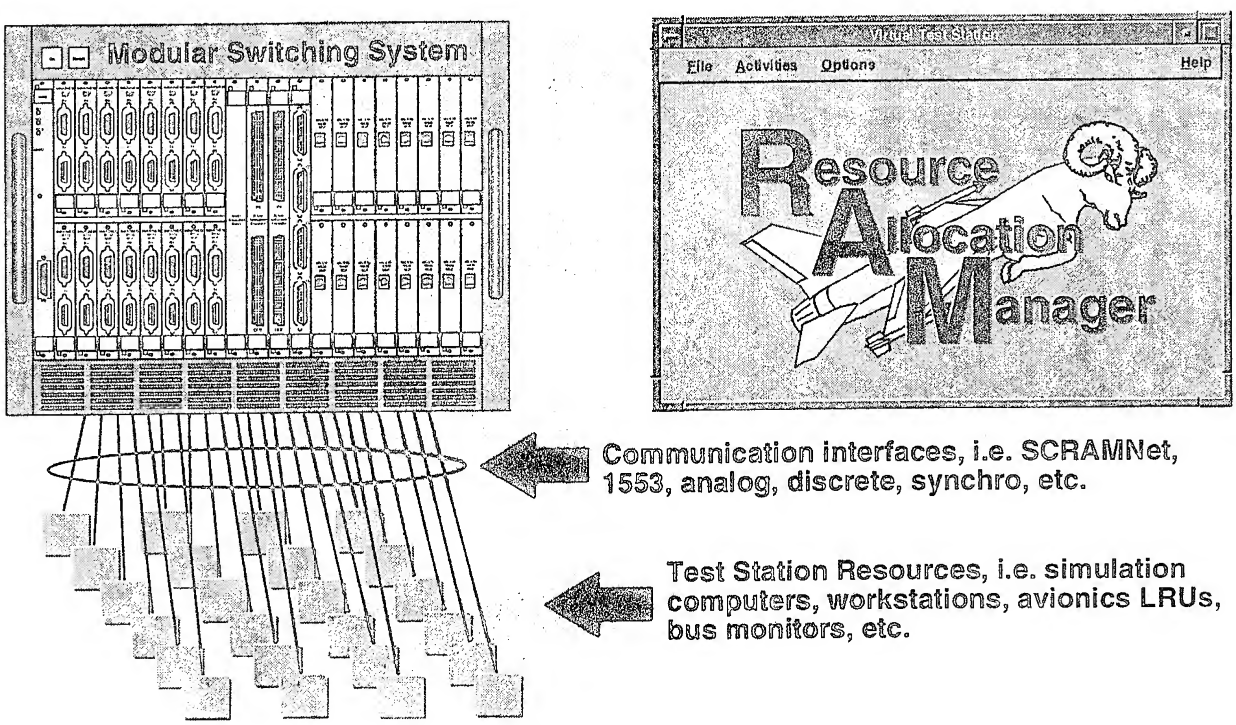


Figure 1, VTS Components

Avionics Testing with VTS

Avionics software testing represents a spectrum of requirements across the testing phases from the simplest static tests with a single LRU proceeding through full dynamic and integration testing to final system testing. Traditionally, we have created completely separate test stations to accommodate the testing in these different phases. With the VTS, we no longer have independent test stations, but rather a large collection of shareable resources that we can combine efficiently to best suit the particular testing phase we are currently in.

The VTS enables test engineers to rapidly reconfigure testing resources during subsystem, integration, and system testing phases. During subsystem testing, the test engineer selects a small subset of resources to confirm basic OFP functionality through static tests. The test engineer progressively adds additional resources as testing progresses into greater levels of dynamic testing. As we transition to the integration test phase, the test engineer conducts full dynamic testing starting with a single LRU and adding more until testing is accomplished with the complete system. Simulated or emulated LRUs are often used in the early testing phases and are later switched out and replaced by actual LRUs. System testing is the last step prior to flight test. OFPs are tested with as much of the actual avionics suite as possible. All of this testing, through each of these phases, can be accomplished in the VTS with a common set of resources that are connected and reconnected in whatever configuration is required at each phase in the testing process. The VTS flexibility also enables a greater number of test engineers to work in parallel during subsystem testing when each test engineer requires fewer resources. Each engineer uses only the resources actually required for their test, leaving the remaining resources available to others.

Background

SAIC is working with WL/AAAF to improve the performance and reduce the cost of embedded software support. We are focused on research, prototyping, and limited production of test station technologies, including high performance distributed simulation architectures, techniques for the effective use of commercial-off-the-shelf (COTS) hardware and software, and techniques for the use of open standards, such as Ada, X/Motif, VME, UNIX, etc. This activity produced the VTS Concept, an evolution of earlier research in distributed, real-time simulation architectures. We are also developing methods and tools to improve the efficiency of OFP testing, such as AutoVal, a toolset that automates the OFP test process.

SAIC's Aeronautical Systems Operation (ASO) has spent the past four years implementing a comprehensive program of software process improvement based on the Software Capability Maturity Model (CMM) from the Software Engineering Institute (SEI). This program of continuous process improvement has led to our assessment by an independent outside agency at SEI Level 3, a level characterized by fully defined software engineering processes and program management practices. Of all the assessments conducted nationally, the SEI reports that only 9% have been assessed at this level or higher. SAIC is conducting the VTS Program with WL/AAAF in accordance with SEI Level 3 processes. It was one of the programs reviewed during SAIC's software process assessment and found to be Level 3 compliant.

VTS Technologies

The technologies we are using to implement the VTS fall into three groups: reuse of existing hardware and software technologies, the RAM software suite, and the MSS hardware. The VTS can be used as the foundation for construction of a new ISF/SIL, or it can be used to supplement an existing facility and greatly enhance its efficiency and cost effectiveness. In an existing facility that has simulation technologies based on a modular, loosely-coupled, distributed processing architecture, VTS technologies can be easily inserted with minimal disruption to the existing system. In such a case, an organization can gain almost immediate benefit from the efficiencies of automated scheduling and allocation of resources.

Even test stations based on a tightly-coupled, monolithic architecture can capitalize on VTS technologies. WL/AAAF has a cohesive architecture and toolset that is well suited for transitioning a large, monolithic simulation system into a modular, distributed system. This architecture consists of an executive, simulation model design standards and templates, and programming guidelines. It also includes a number of support tools for simulation control, automated configuration management, automated testing, etc. The executive and model design standards are the mechanisms for making the simulation software modular and independent in a distributed processing environment. The programming guidelines focus mainly on ensuring software portability across different processor types and on providing high-performance, deterministic real-time operation. WL/AAAF successfully used this approach to transition a large, mainframe-based simulation for the F-16C/D to a distributed system consisting of COTS single-board computers, minicomputers, and high performance workstations. The entire effort took less than a year for more than 65,000 lines of simulation code. The simulation models were inherently modular and adapted easily to the distributed architecture. The most time consuming effort was conversion of non-ANSI standard language constructs due to the mainframe's non-standard compiler. This experience graphically illustrated the benefits of designing systems that are hardware vendor-independent and based on open standards.

Resource Allocation Manager (RAM)

The RAM is responsible for management of computational and other resources in the VTS. It is written entirely in Ada and is designed for operation on a standard UNIX workstation utilizing a windows-style, X/Motif graphical user interface. The RAM can run on any workstation in the facility that can reach the MSS over Ethernet. Figure 2 shows a system block diagram of the VTS, and how the RAM and its various components interact with the MSS. We have established a client-server relationship with some of the RAM components to permit the RAM to be used by more than one user at the same time (e.g., from the workstations at each test engineer's desk).

The RAM is designed to serve the needs of three classes of users: test engineers, system maintainers, and the system manager. Test engineers use the RAM to configure and schedule test sessions. System maintainers use the RAM to add or remove computing or avionics resources, to update the simulation software library, to assist test engineers, and to handle problem reports. The system manager uses the RAM to monitor resource utilization, to set scheduling priorities,

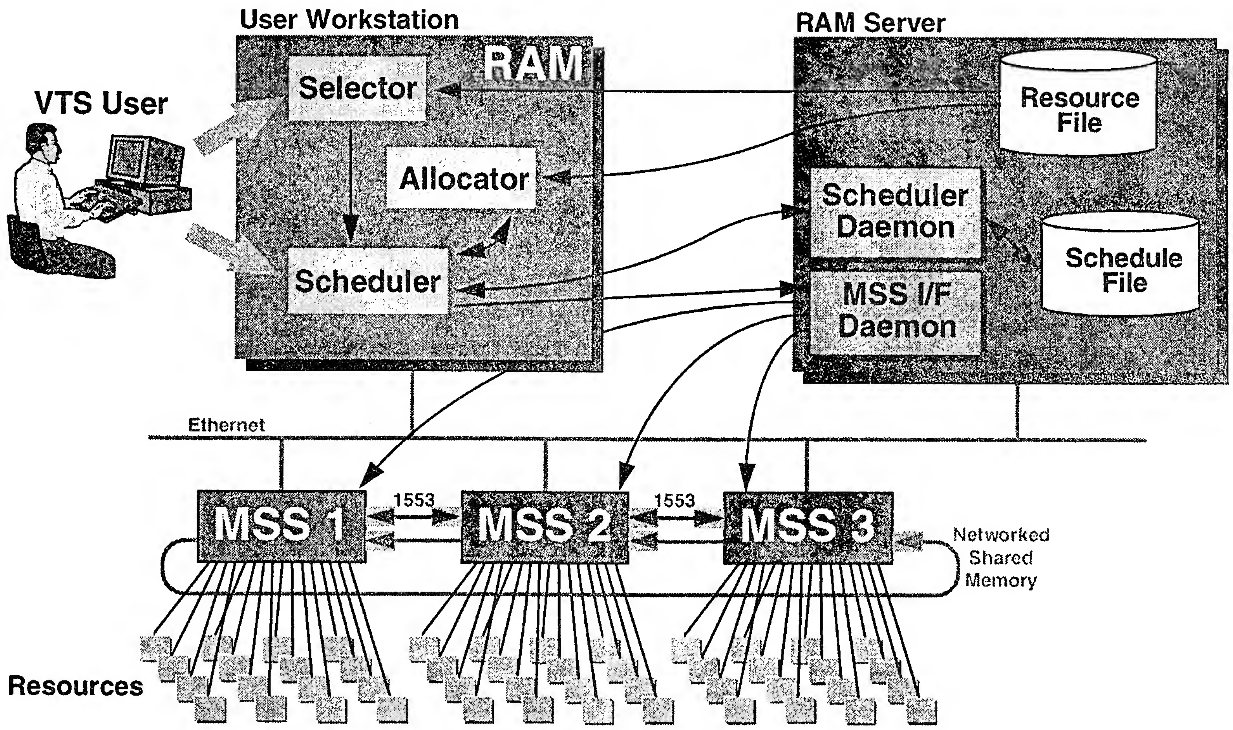


Figure 2, VTS System Block Diagram

and to adjust the resource mix. The most prevalent users are the test engineers, and the RAM has been carefully designed to streamline their access to the system for testing embedded software.

RAM Operation Overview - The Test Engineer

Test engineers first use the RAM to select the items they need for a test session. The items they select are not physical test station resources, but are the simulated or actual components of the avionics system they need to perform the particular test activity. As they configure the simulation components, they are, in effect, selecting a set of simulation software. They do this, however, without regard to what physical computing resources will be needed to run that software. The entire process of mapping the selected components to specific software and then mapping that software to computing resources is handled automatically by the RAM. As a result, test engineers don't have to become experts on all the configuration and loading rules for dozens of computers. They can concentrate on being experts on testing embedded software.

After the test engineer has selected a configuration for a test session, he uses the RAM to schedule the session. The availability of resources to support a test session is a function of what resources other test engineers are using at any given point in time. When the test engineer activates the RAM scheduling function, the RAM presents the test engineer with a graphical display of the time periods during the next week when sufficient resources are available to support his test session. The test engineer then schedules one of these open time slots, and the RAM reserves the configuration he has selected at that time.

Later, when the scheduled time arrives, the RAM makes a quick check to see that the required resources are still available. If they are, the RAM directs the MSS to connect the resources, downloads the selected software to the proper resources, and alerts the test engineer that the test session is ready to begin. At the end of the session, the RAM removes the downloaded software and signals the MSS to break the resource connections.

RAM - Selection

The first level of selection is organized by "block" (Figure 3). This categorization is purely arbitrary and is based on how the Air Force upgrades aircraft. A block simply represents a collection of software components that can operate together for the support of a particular aircraft, some of which may be common to more than one block (or version of aircraft). The "block" nomenclature is data driven and may be tailored to the way systems are organized in the particular facility. In a facility that supports more than one aircraft, this first level of selection could apply to those different aircraft. Once a block is chosen, the test engineer is prompted to select a test session configuration, either one that he has previously stored, or a standard system configuration template.

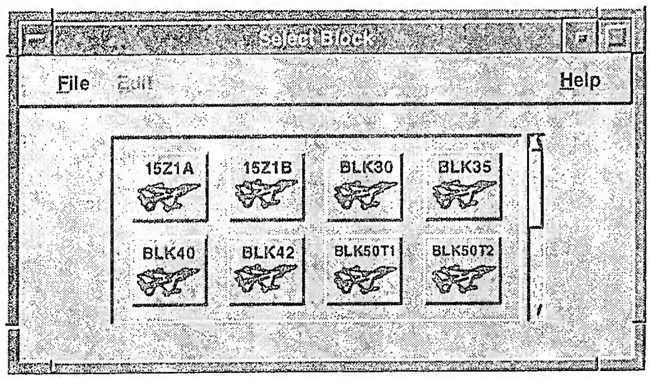


Figure 3, Block Selection

Figure 4 shows the main RAM selection window that the test engineer uses to specify the configuration of a test session. The test engineer simply clicks on the button for each avionics component he needs for this test session. If a particular component has more than one option available, as in the case of models with differing levels of fidelity or features, a pop-up window appears and prompts the test engineer to select the desired option. Everything in the selection window is data driven and can be tailored to the specific facility environment. The particular partitioning shown here, both by group and by model, pertains to a particular

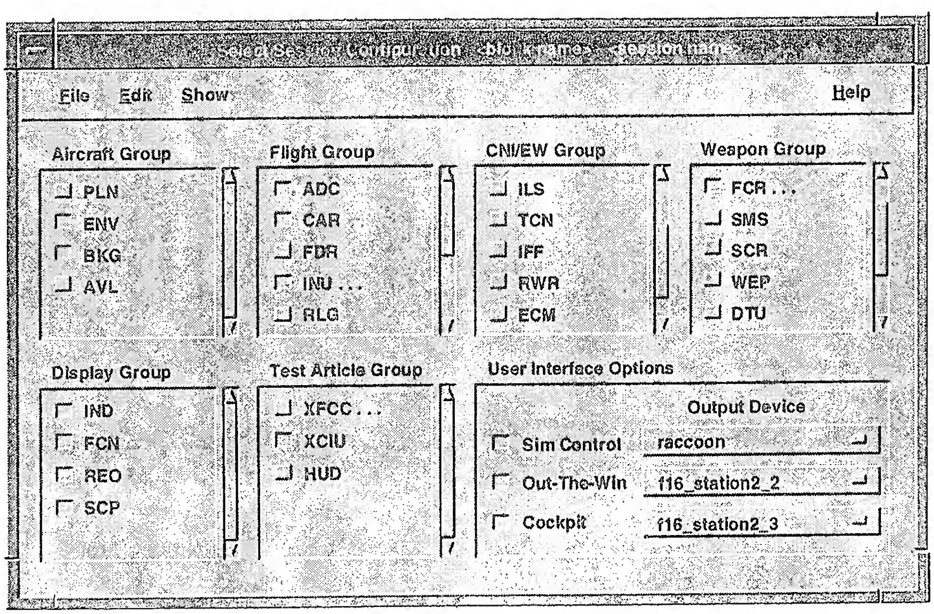


Figure 4, Session Configuration Selection

F-16 facility and would be different for different facilities. To get maximum benefit from the VTS, the simulation software models should be both modular and independent.

RAM - Scheduling

After selecting the desired configuration for his test session, the test engineer opens the Scheduler window to schedule the session (Figure 5). It is possible to open both the Selector window and Scheduler window up at the same time, and in fact, this is the preferred approach. This permits the test engineer to adjust the test session configuration in the Selector window and quickly see the effect on available session times in the Scheduler window. In this window, color is used to indicate availability. Red means that the resources needed to accommodate the configuration selected are not available. Green means that they are available, and blue shows other test sessions already scheduled by this test engineer. The blue button superimposed on the schedule bands represents the test session. The test engineer uses the workstation mouse to drag this button to a desired time slot within a green band. He can also expand or contract it to adjust the desired session length.

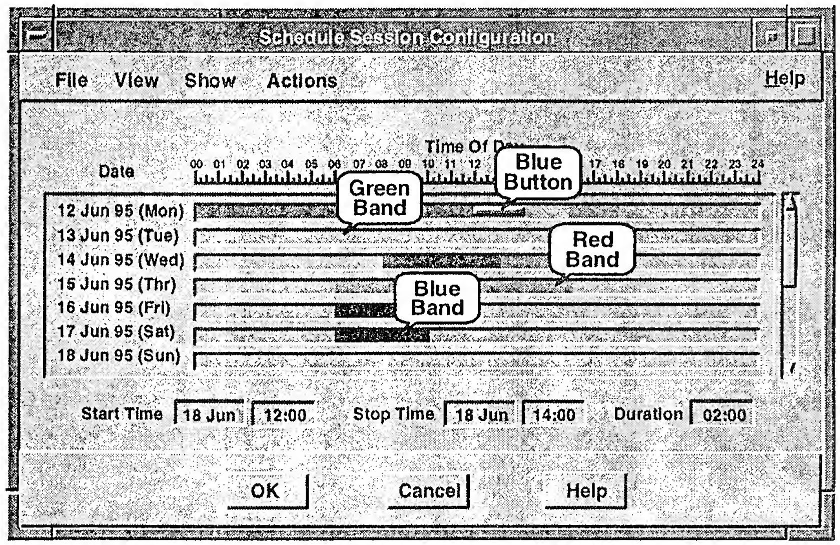


Figure 5, Session Scheduling

RAM - Allocation

Automatic allocation of resources by the RAM is the real power of the VTS Concept. The RAM takes the software selected by the test engineer and, by analyzing the resources that are required and the resources that are available given the current schedule, it selects the best combination of hardware resources. This also guarantees that the same resource won't be scheduled by two users at the same time. After the hardware resources have been allocated, the RAM then computes the best load-balanced distribution of the software on the allocated resources. This is a very sophisticated algorithm and was considered to be one of the higher technical risks when we began the development of the RAM. Through rapid prototyping, we experimented with several alternative approaches and produced a successful solution that works very well.

RAM - Maintenance and Diagnostics

The RAM maintenance component is used by the system manager and system maintainers to establish and maintain the data files that define the facility and all the resources controlled by the

VTS. This includes adding and removing resources from the facility, and also taking them on and off-line, i.e. changing their availability for allocation to test engineers. If a resource fails while a test engineer has it allocated, the test engineer may also take the resource off-line. Following successful completion of a maintenance activity to repair it, a system maintainer then puts the resource back on-line. The RAM maintenance component continuously accumulates usage statistics for all of the resources. These show the system manager which resources are underutilized (and may potentially be removed) and which resources are in the highest demand (and may be supplemented to increase organizational throughput). The system manager and system maintainers also use the RAM maintenance component to log and track problem reports on the system.

The RAM diagnostics can be thought of as a system-level built-in test (BIT). At periodic intervals set by the system manager, the RAM automatically allocates unused resources, and then downloads and runs self-test software. If a test fails, the resource is taken off-line and the system manager alerted. If the test passes, the resource is deallocated and remains available. For example, twice a week, an avionics computer resource that is not currently allocated could automatically be powered on and loaded with a diagnostic program. In conjunction with the diagnostic program, an automated test could execute to verify correct operation of the hardware. If the diagnostic fails, then the resource would be automatically taken off-line for maintenance. If it passes, the avionics computer would be powered off and remain available to test engineers for future test sessions.

Modular Switching System (MSS)

The MSS has a standard VME 6U form factor chassis and accommodates up to four switch modules and up to 32 modular communication plug-ins in any combination (Figure 6). Communication with the RAM and control of the switching circuitry is handled by software in a COTS VMEbus single board computer in the far left slot. The four center 6U slots accommodate switch module plug-ins that can be mixed and matched to adapt the MSS to the requirements of the particular ISF/SIL. The 32 remaining 3U slots are used for Modular Communication Plug-ins (MCPs) that are compatible with an installed switch module. Any MCP-type can be installed in any slot.

The RAM keeps track of which resources are connected to which MCPs in each MSS. Communication with the MSS from the RAM is handled via TCP/IP over Ethernet. The embedded controller in the MSS receives connect and disconnect commands from the RAM, coordinates with the other MSSs (if present), and activates the appropriate switches.

There are currently three switch module types. Crosspoint switch modules are used for high-speed fiber or twisted-pair networks with a ring topology such as SCRAMNet and SMARTNet. Linear switch modules are used for lower speed networks with a bus topology, such as MIL-STD-1553 and Ethernet. Our design emphasis throughout has been on complete modularity, flexibility, and reconfigurability. The use of plug-in switch modules and MCPs allows us to tailor the MSS configuration to the unique requirements of any avionics test facility.

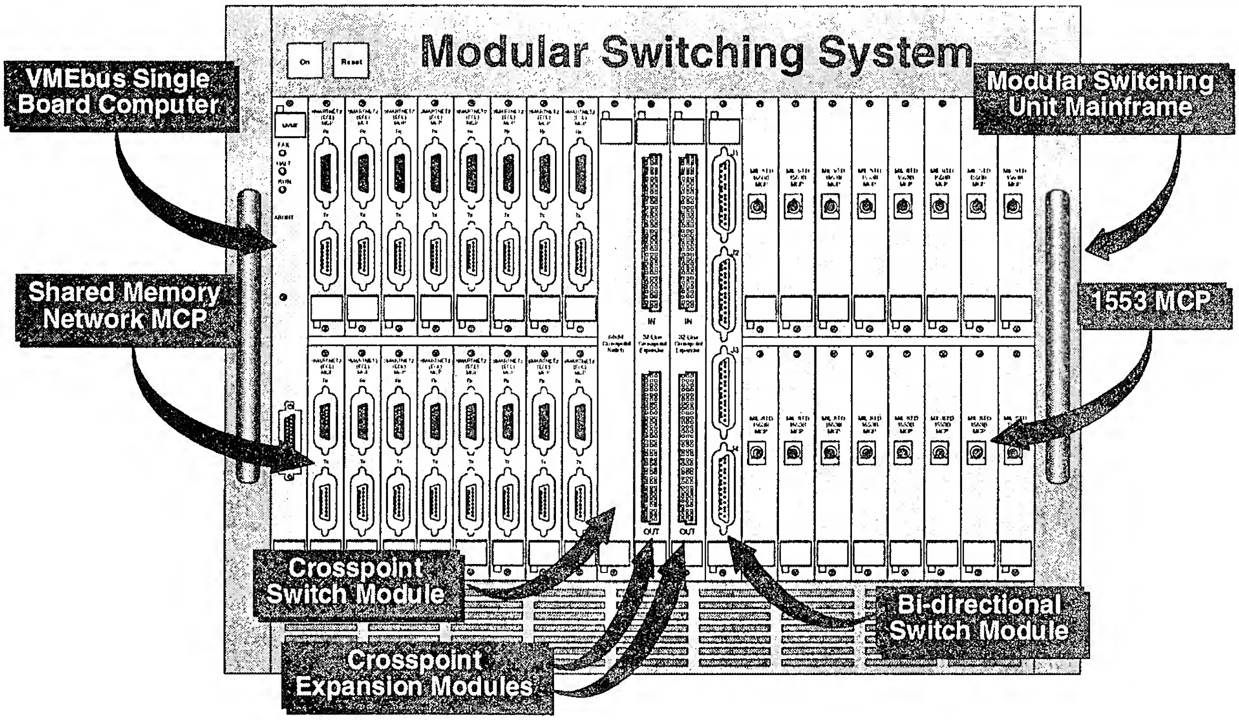


Figure 6, Modular Switching System

A variety of MCP-types are provided. The MCP design is also completely modular and new MCP-types may be developed to handle additional signal types and/or emerging network technologies in the future without having to replace the entire MSS.

Conclusion

The VTS concept represents a very powerful extension of present ISF or SIL architectures for all levels of avionics software testing. It capitalizes on the modularity of the modern, high performance distributed processing paradigm to provide a complete system of automatically reconfigurable test station resources. This total reconfigurability helps optimize resource allocation, enhances maintainability, and increases work efficiency by providing exactly the capability each test engineer needs for each phase of testing. Most importantly, the VTS concept can reduce the cost of the avionics support facilities for large, complex weapon systems. Because of better resource utilization, VTS reduces the number of resources required which lowers both facility acquisition costs and facility life cycle (support and maintenance) costs. Finally, VTS reduces avionics support costs by improving testing speed and efficiency. Given the projected cost of support facilities for the weapon systems than are scheduled to come into the Air Force inventory in the next five years, VTS could save the Air Force almost $1 billion.

METHODS AND PROCESSES SUB-WORKING GROUP (MPSWG)

Co-Chairs:
Larry Gore and Larry Skiles

FLEXIBLE, INEXPENSIVE INTERCOM SYSTEM DESIGNED TO MEET THE NEEDS OF THE TEST COMMUNITY

Timothy B. Bougan*
Michael L. Baumgartner

Science Applications International Corporation
EC Test Support Division, Panama City, FL
904-784-1799

1. Introduction

Testing avionics and military equipment often requires extensive facilities and numerous operators working in concert. In many cases these facilities are mobile and can be set up at remote locations. In almost all situations the equipment is loud and makes communication between the operators difficult if not impossible.

To facilitate communication, most test sites incorporate some form of intercom system. While intercom systems themselves are a not a new concept and are available in many forms, finding one that meets the requirements of the test community (at a reasonable cost) can be a significant challenge. Specifically, the test director must often communicate with several manned stations, aircraft, remote sites, and/or simultaneously record all or some of the audio traffic. Furthermore, it is often necessary to conference all or some of the channels (so that all those involved can fully follow the progress of the test).

This paper describes the philosophy and design of a sixteen channel intercom system specifically intended to support the needs of the military test community.

2. Requirements

For our initial design, we attempted to encapsulate as many operational requirements as possible. We foresaw the need to allow flexible communications between operators, radios, and recording devices. We also projected a requirement to connect the intercom to external, non-system devices. To these ends, we specified the following initial requirements:

1) The system needed fifteen stations (four "manned" or "operator" stations, four VCR stations, three radio stations, two generic stations, and two expansion stations). See figure 1.

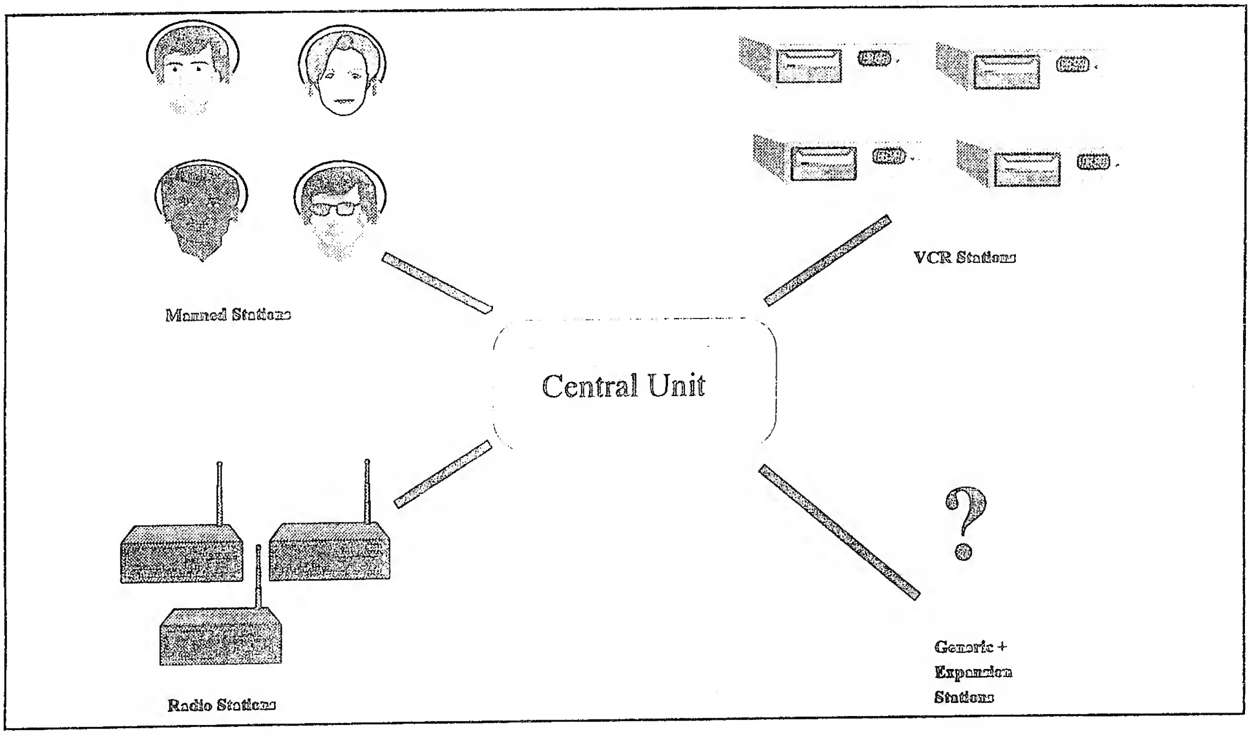


Figure 1 -- Intercom Station Requirements

2) Operator stations need headset jacks and external speakers (with volume controls). The headset jacks include microphone connections. The stations also need keypads so the users can configure them to their communications requirements.

3) VCR stations interface with video recorders used to record communications during tests and operations. These stations require keypads (to select which sources to record) but do not need headset jacks, external speakers, or microphones.

4) Radio stations interface to either VHF or UHF radios. These stations need circuitry to key the radios when a connected station keys its microphone and must also "blank" all out-going audio during transmission. Radio stations do not need keypads, headset jacks, external speakers, or microphones.

5) Generic stations are intended to receive or send audio to/from non-intercom devices. These stations have only audio in and out lines and have no controls.

6) The system must be completely non-blocking and fully programmable.

7 Complex configurations (set by users) must be easily saved and restored.

8) Frequency response must be 100-7000 Hz (or better) for audio communications.

9) The system must support multiple "nets" (or conferenced communications).

3. Approach

We quickly decided that a fully non-blocking system would best be implemented with centralized mixing. That is, incoming audio from each station is appropriately mixed at a central location (rather than in each remote station). This allows us to have a single send and receive pair of audio lines to each station and dramatically reduces the cabling requirements.

Deciding how that audio is to be mixed poses another problem. While some parameters can be specified at system startup (in our case in a configuration file), the operators can dynamically change their configurations with their keypads. We decided, therefore, that each remote station would have a micro controller to communicate with the central unit and control the local keypad and lights. The communication to the central unit would be serial (RS-422, 9600 baud).

The central unit would be responsible for monitoring all remote station communications and mix all incoming audio to generate the correct outgoing audio (for each station). To do this, the central unit needs digitally-programmable analog switches to properly route each incoming audio line to specific summing circuits. The switches must be robust enough to allow any combination of incoming signals to be combined and sent to any station. The central controller needs to be capable of translating all the received communication (from the remote stations) into appropriate commands to program the switches.

These design decisions gave us complete, on-the-fly flexibility with respect to which audio signals will be routed to each station. We could easily implement a fully configurable and non-blocking intercom... and easily implement the conferencing capability (nets).

We further decided to design a single remote station circuit board that could be easily configured as any of the three primary stations (manned, VCR, or radio). Most of the circuitry is identical from station to station, and a few judiciously placed jumpers let us simplify manufacturing dramatically by limiting the number of unique components.

Meeting the frequency response requirements is not difficult with the abundance of proven, off-the-shelf audio circuits available today. Our approach here was to use very simple, text-book circuits to mix and amplify the audio signals. We also decided to transformer couple all audio input and output stages to eliminate line noise and 60 cycle hum.

4. Design

4.1. Central Unit

We chose to use the Analog Devices AD75019 (16x16 Crosspoint Switch Array) to route all incoming audio signals to the proper summing circuits. This part allows any of sixteen input signals to be routed to any of sixteen output pins, and can be programmed on-the-fly by a serial stream of bits, a shift clock, and a programming strobe. The input signals are first conditioned by a simple amplifier circuit (using a TI TL084 operational amplifier in an inverting configuration). The output signals from each crosspoint switch feed a 16-input summing circuit (also implemented with a single

TL084). See figure 2 for an example. For sixteen stations, we need sixteen switches and sixteen summing circuits.

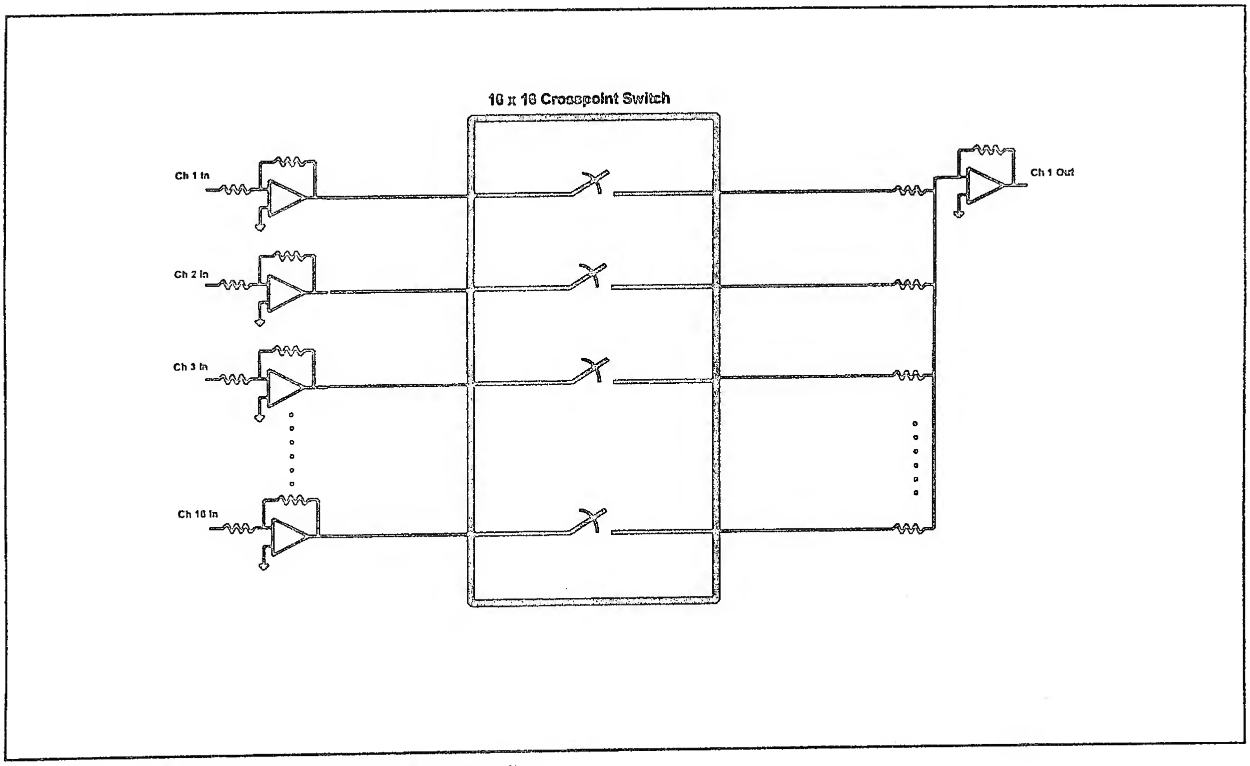


Figure 2 -- Audio Mixing (One Channel)

The central unit also has sixteen RS-422 transceiver chips to communicate with each remote station. To simplify the circuitry, we decided to "multiplex" the serial communication for each remote station. That is, the central controller need only have a single serial port and the system board must have circuitry to select which station is connected at any given time.

By far the most difficult task for the central unit to tackle is correctly programming each switch for the almost limitless combination of connection possibilities. To ease the task, and to keep costs very low, we opted to use an embedded PC for the system controller. This allowed us to develop the controller code on a desktop PC (using Borland Pascal). A basic PC motherboard (386 class) costs less than $130, and even with all associated hardware the total cost of the controller is less than $500.

Since the PC already has a serial port, we added an RS-232 transceiver to the system board. We also decided to use the PC's parallel printer port to provide high-speed data to the system board (for the purpose of programming the crosspoint switches and selecting with remote station to communicate with). See figure 3.

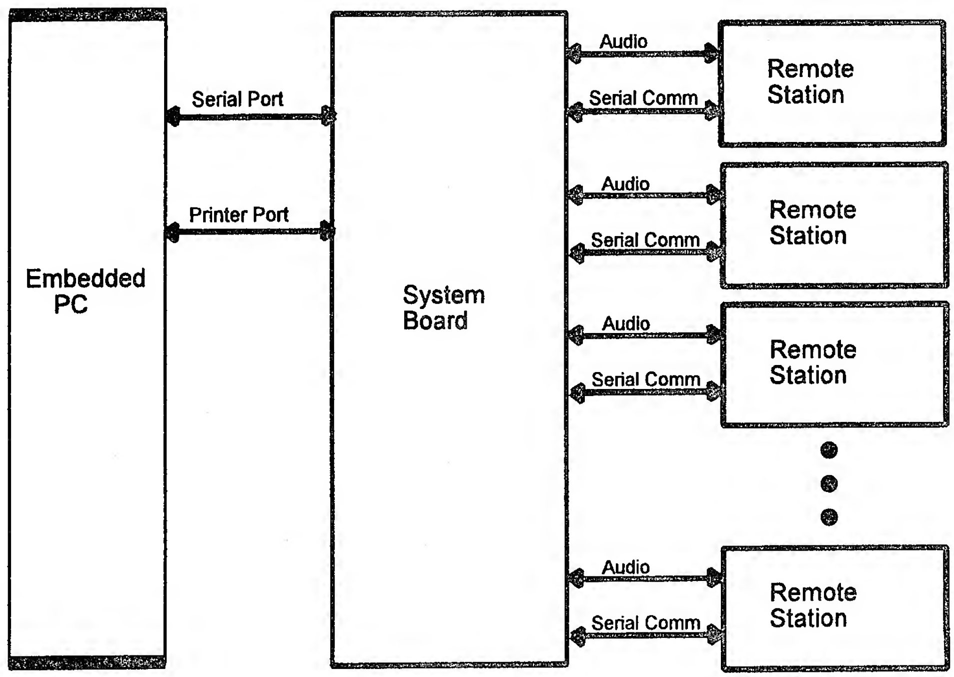


Figure 3 -- Intercom Block Diagram

The last task we had to tackle on the system board was how to translate the PC's parallel commands to program each switch and select each remote station. Here we used a single Xilinx 4005 (which is a field programmable gate array or FPGA). This particular part has 112 I/O pins, 616 internal flip-flops, and approximately 5000 logic gates. Using a schematic capture package (Viewlogic's Workview), we were able to easily build the required glue logic in one chip. The FPGA converts the parallel data into a serial stream to program each switch and contains a state machine that generates the necessary clock and programming strobes. The FPGA also multiplexes the serial communications data based on a station number provided by the PC. Also, because the part is easily programmed, and because we used less than 20% of its available capacity, we've left ourselves with considerable room for adding features and capabilities in the future.

All analog signals to and from the central unit pass through 600 ohm coupling transformers, and we adjusted the signal levels to the industry standard of 1 VPP max.

4.2. Embedded PC

Although we could have bought a true "embedded" PC, we opted to purchase motherboards intended for desktop systems. They are readily available and usually one-tenth the price of embedded systems. We bought two 3.5" drives for the PC... the "A" drive is enclosed in the case while the "B" drive is accessible from the front panel. The A drive holds the boot floppy disk and automatically loads the intercom's executive program. A configuration file (located on the boot floppy disk) sets up the initial parameters.

Once the intercom's executive program is running, the intercom is operational. At any time the operator can "save" the current configuration (including all connections selected on the keypads) by pressing a button on the front panel. The configuration is saved as a binary image on drive B (the accessible drive).

The intercom's executive program continually "polls" each remote station (10 times/second). The stations return their status, including which keys (if any) had been pressed. The program then calculates the correct crosspoint switch patterns are necessary to implement the function and reprograms the appropriate devices. The program also sends instructions back to the remote station (to light buttons, change button colors, or key the radios).

4.3. Remote Stations

Each remote station circuit board contains a microphone interface. By removing a single jumper, the microphone is disabled and the board can accept input from a radio or VCR. The audio is conditioned to 1 VPP (max) and transformer coupled before it leaves the remote station box (bound for the central unit).

Audio coming from the central unit is also transformer coupled. Each remote station board has all the necessary components to drive a headset speaker and an external speaker. Each speaker has a separate volume control.

The heart of the remote station is an Intel 8751 micro controller. The 8751 runs a tight loop in which it scans the 18-key keypad, multiplexes the keypad LEDs, and handles all the serial communication from the central unit.

As the 8751 scans the keypad, it debounces the keys and prevents run-on and roll-over. It subsequently lights the individual LEDs by multiplexing the LED control lines (only 6 of the 18 LEDs are actually on at any given time). The LEDs are two color, but by rapidly changing current direction, we get four states: OFF, RED, GREEN, and YELLOW. It is up to the central unit to decide which LEDs are to be lit and what color they are to be... the remote station is acting primarily as a "dumb terminal".

The 8751 also senses the state of the microphone key and passes that information to the central unit. It can also close a relay (which causes the radio to "key") when so commanded by the central unit.

5. Conclusion

SAIC built and delivered two of the systems discussed above to the Office of the Test Director.

We were successful in meeting our original design specifications. The SAIC system, now named *FlexComm I*, is fully non-blocking and completely programmable. It integrates seamlessly with radios and recorders. The system is robust in that we built in a lot of error and status checking as well as integrated "watchdog" timers to monitor CPU operation. All of the audio lines are transformer coupled, and the digital serial communication lines are all differential.

FlexcComm I also has a number of features difficult to find in off-the-shelf systems. For instance, operators can select to talk and listen (on a conference net) or only to listen. Also, microphones can be configured to require keying or may be set as "hot". Furthermore, the systems may be "bridged" (essentially "linking" two or more FlexComm systems together). These connections allow communication between workgroups, while still retaining control and security within each local workgroup.

FlexComm I has excellent frequency response (50Hz-10KHz) and is reasonably inexpensive to build (about $1.5K/station). All in all, FlexComm I is a near perfect match for the requirements of the test community.

Automating the IV&V of RWR Operational Flight Programs and Mission Data Bases

C.K. Cole*
J.F. Corbett

Georgia Tech Research Institute
Atlanta, Georgia 30332-0829

Abstract

Because of the vast improvements in chip technology, today's avionics systems are being designed with more memory and greater programmability than was once thought possible. A typical avionics system going to the field ten years ago had a memory capacity of 64 to 128 kilobytes (KB) with hardware that provided little if any programmability of system functions. The avionics systems of today commonly employ 1 megabyte (MB) or more of memory and often include so-called "register-programmable" devices that define much of the hardware.

In the case of a radar warning receiver (RWR), this tremendous memory expansion has provided the opportunity for the writers of RWR Operational Flight Programs (OFPs) to include many new software capabilities. It has also provided the opportunity for writers of RWR Mission Databases (MDBs) to include threat identification parameters for many additional threats in the MDB.

Because the OFPs are more complex and because MDB writers often add but rarely delete threats, there are many more software algorithms to test and many more threats to simulate and run against an RWR under test. Thus, the independent verification and validation (IV&V) process has become much more demanding, and some automation of the process is necessary to ensure adequate system testing within a reasonable budget.

The traditional method of verifying an RWR MDB is to first have a threat generator output a signal that simulates the signal of a threat system programmed in the MDB and then to have a test engineer visually check the output display of the RWR and make a notation of the test results. This process is repeated for every threat and threat mode programmed in the MDB, which could be 500 or more simulator files. The test takes days to accomplish and provides little or no system performance statistics other than correct identification (ID) of the threat.

An automated test process accomplishes several objectives. First, the entire one-on-one test phase can be completed in a timely fashion, and preferably overnight; this makes the test process less costly and less time consuming and frees the expensive threat generators for other tasks during normal working hours. Second, because automated tests can be run more quickly and economically, it encourages complete retesting after an OFP or MDB change. Third, the automated test can generate many system statistics such as response time, direction finding (DF) accuracy, correct ID, etc. Fourth, the automated test is more repeatable and less prone to human error.

This paper discusses the process of automating some phases of the Radar Warning Receiver system validation. This process begins at the start of hardware design and continues through the development of the automated test software that controls the running and scoring of the test.

Introduction

Radar warning receivers (RWRs) are designed to search for signals from hostile radar systems and to warn an aircraft pilot of the presence of those systems. RWRs can detect signals from all types of radar systems (search, acquisition, target tracking, missile guidance, etc.). Once the signal has been identified, the aircrew is given a warning signal, usually both aural and visual, and an approximate indication of the location and type of the threat.

RWRs were first developed during World War II and were originally only used in bombers. During the Vietnam War, the North Vietnamese used radar-guided surface-to-air missiles (SAMs) against US Air Force and US Navy fighters with astounding success. The resulting losses triggered the US Air Force and US Navy to develop countermeasures to these threats. Thus began the proliferation of RWRs into all types of military aircraft.

The early RWRs provided no detailed threat identification. Instead, the RWR simply monitored a range of frequencies and displayed the bearing of the detected signals on a cockpit-mounted display. Although the technique was simple, it worked very well. Within months, a combination of RWRs and Wild Weasel anti-radar strikes brought the SAM kill rate down from 50% to 3%.[1]

Modern RWRs provide detailed threat identification to the aircrew. Detailed threat identification is important because it allows the aircrew to decide what action, if any, is necessary. If the hostile radar is a missile tracking radar or a radar on an airborne interceptor, then the crew must immediately begin maneuvers to defeat the threat. On the other hand, a long-range surveillance radar may require no action at all.

Threats are identified by comparing parameters such as frequency, pulse repetition interval (PRI), pulsewidth, scan pattern, and scan rate against a database of all known threats as reported by intelligence sources. The computer software that performs the threat identification process is known as the Operational Flight Program (OFP) and the database that contains the threat parameters is known as the Mission Database (MDB).

As recently as 10 years ago, a typical RWR might contain only 64 to 128 kilobytes (KB) of read-only memory (ROM). This severely limited the size and complexity of OFPs and MDBs. Software updates were also costly because the ROM chips could not be reprogrammed onboard the aircraft. A typical RWR in the field today contains 500 KB to several megabytes (MB) of electrically-erasable programmable read-only memory (EEPROM). This tremendous memory expansion has allowed OFP writers and MDB writers to add sophisticated software algorithms to counter the hostile threats in the world today. Also, because EEPROM is easily programmable onboard the aircraft, the aircrews now demand rapid OFP and MDB updates.[2]

The combination of more complex OFPs, larger MDBs, shorter schedules, and slimmer budgets adds a new dimension of difficulty in designing an adequate test process. However, by using technologies available today, testing and independent verification and validation of RWR software can be accomplished in a thorough manner within schedule and budget constraints. This paper describes techniques that systems, hardware, and software engineers should consider when attempting to improve system testability.

RWR Testability Issues

There are numerous issues that are of concern in determining the testability of an RWR. Of these, data visibility and OFP intrusion are the two most important. The following defines these terms as they will be used throughout this paper.

"Data visibility" refers to internal OFP variables being conveniently available external to the RWR for electronic recording and manipulation. If internal OFP variables (or files) are not visible outside of the RWR, then the RWR will be difficult, if not impossible, to test. Traditional RWR testing has relied on the RWR display as the sole data source for determining test results. Because of the additional complexity of today's RWR OFPs and MDBs, this method is no longer sufficient. With current technology, much more data visibility is easily achievable.

"OFP intrusion" refers to the processing cost of dumping OFP data to an external interface. It is desirable to make internal OFP files observable to the outside world without any impact to the normal operation of the OFP. Any software action that is not directly related to the interception, identification, and display of a hostile radar system is "intruding" upon the normal RWR OFP operation. Obviously, some processing time is required to dump OFP files to the external interfaces, but designers can minimize this impact.

Hardware Design Issues

Thorough and efficient testing of RWR software is possible if the following rules are observed during the hardware design process.

- Outfit the system with a high-speed external interface.
- Provide connectors to the high-speed interface so that it is easily accessible from the outside of the box housing the system.
- Design an embedded processor that is dedicated to servicing the high-speed interface.

Providing a High Speed Interface

There is at least one RWR that was fielded during the Vietnam War that is still in use today. This RWR must use an RS-232 interface for dumping OFP files to external devices.

Because the RS-232 interface is so slow relative to the speed of the embedded processor, the RWR display goes blank while data is being dumped. This happens because the normal operational timing of the RWR is totally disrupted by the dumping of test data.

Most RWRs going to the field in the 1990s include one or more MIL-STD-1553B data busses. MIL-STD-1553B (1553, for short) is a high-speed (1megabit/second) serial data bus protocol. It is typically used to support integration of EW devices by allowing these devices to communicate with each other. Because of this communication medium, the various EW devices can act in a concerted manner to defeat a hostile threat. The 1553 bus is also a suitable interface for dumping OFP data for test purposes.

Providing Connectors to the High-Speed Interface

Once the RWR is outfitted with a high-speed interface, it should be provided with connectors that allow for easy and inexpensive hookup to test devices. Any interface that is of a custom design or requires custom connectors or cabling should be avoided. A ready solution to the problems of connectors is provided by MIL-STD-1553B; 1553 data bus couplers and cables are readily available at a reasonable cost. The use of standard 1553 connectors also provides direct access to the physical RWR 1553 bus wiring, since the 1553 is so often used to allow the RWR to communicate with external EW devices.

Dedicating a Processor to the High-Speed Interface

In order to minimize the software intrusion on the master RWR OFP, the high-speed interface should be designed with a microcontroller dedicated to the servicing of the interface. This increases the cost of the RWR, but it will provide a measurable performance improvement. By allowing the master OFP to be concerned only with sending data to and getting data from this processor, the test data transmission should take no more than ½ of 1% of the total processing time of the master OFP.

If the only foreseen usage of the 1553 bus is to dump data for test purposes, then the dedicated processor may be omitted. However, such an omission may be a very shortsighted decision if one considers the future of EW systems. Integration of EW systems, usually via a 1553 data bus, is becoming more and more common, and in integrated systems, the message traffic on the 1553 interface can be very heavy. Thus, a processor dedicated to the servicing of the 1553 bus is a very good long-term investment.

Interface Design Issues

Whether the RWR has been outfitted with a 1553 data bus or some other high-speed interface, there are several interface rules that must be followed in order to enhance system testability. These rules are listed below.

⇨ Allocate interface resources for dumping of internal OFP files.

⇨ Design multipurpose interface messages that will support both EW integration and OFP testing.

Allocating Interface Resources

An interface control document specifies the message formats and message transmission rates for an interface. It is natural and important that interface messages necessary for operational performance get top priority when interface resources are allocated. However, in order to ensure that RWR systems are maintainable and testable, it is important that interface resources are allocated to provide for the transmission of internal OFP files that have no real operational value to the system.

In the case of 1553, this means allocating 1553 bus bandwidth and 1553 subaddresses for test messages. Depending on the system, 1553 subaddresses are typically not a problem compared to bus bandwidth. Many 1553 busses have very heavy message traffic and many persons are reluctant to squeeze in messages that obviously have no operational value. However, these same persons are usually the first to demand rapid OFP and MDB updates. The user community must be convinced that the availability of internal OFP files on the 1553 data bus is a key issue in decreasing the amount of time required to complete a software update. These test messages dramatically increase the observability of RWR data, which subsequently increases the productivity of OFP testing and IV&V.

Designing Multipurpose Messages

Many interface messages can be used for both EW integration and OFP testing if the messages are formatted appropriately. For example, one of the most common RWR OFP files is the Emitter Track File (ETF). An RWR ETF record will contain hundreds of data elements that store known information about a threat being tracked by the RWR. It is common for an RWR ETF record to require 128 to 256 16-bit words to store all data relevant to a threat.

In all integrated EW suites known to the authors, an RWR ETF message is part of the 1553 data bus traffic. The other devices on the 1553 bus (jammers, chaff/flare dispensers, etc.) use data elements from the RWR ETF record to select an action to take against that threat. While an EW device may be interested in one subset of data elements, a person debugging an OFP may be interested in another. The former may use a portion of the ETF to determine when to release a flare, while the latter may use a different portion of the ETF to determine the ID of a threat.

When adding test-specific data to the 1553 bus traffic, the interface designer must be careful to avoid duplication; the 1553 bus bandwidth is too precious and such duplication is unnecessary. All 1553 messages should be formatted such that the message is suitable for use by all parties interested in the message: EW devices on the aircraft 1553 bus, OFP debugging tools, and automated IV&V tools. Also, since 1553 messages are only 32 words in length, an entire ETF record cannot be transmitted in one message. Therefore, the interface designer is challenged to include all of the data elements needed by all destinations into one 32-word message.

Software Design Issues

Humphrey states, "Ensure that testability is a key objective in your software design."[3] There are many rules that OFP writers can follow to improve the testability of RWR software, but the following are the two of the most crucial.

- ⇨ Define the most critical OFP data elements for output to the high-speed interface.
- ⇨ Minimize intrusion on the OFP when dumping test data.

Selecting OFP Files to Output

Whatever external high-speed interface is used for dumping data, it is very unlikely that the OFP writers will have the resources available, either in processing time or interface bandwidth, to dump every OFP data element that would be useful for debugging. Therefore, OFP writers must select a limited subset of OFP data for sending out of the RWR for OFP debugging and validation. This selection must be done early in the project and must be done accurately, since interface designs are much more painful to change than are software designs.

Minimizing OFP Intrusion

The OFP writers must make every effort to spend as little processing time as possible to send OFP data to the high-speed interface. Minimizing OFP intrusion is important for several reasons. First, every second that is spent transmitting messages is a second that is not spent identifying potential threats to the aircraft. Second, ensuring minimal interference with normal OFP operation means that OFP operation in the laboratory will more accurately represent field operation.

One simple way to minimize the processing time required to output OFP data is to format the OFP data in the same manner as the message traffic on the high-speed interface bus. For example, if a 1553 bus is used to dump Emitter Track File records, the software engineer should format the first 32 words of the ETF record identically to the format of the ETF Record Message on the 1553 data bus. If this is not done, then the OFP writer will have to write a subroutine that reformats the ETF record to that of a 1553 message on the fly, wasting precious processing time.

Another way to minimize OFP intrusion is to design semaphores into all the messages that allow the dumping of data to be disabled when a message is not being transmitted on the external bus. For example, if the OFP has an internal file that is critical to transmit over the bus for OFP testing but is not useful in an operational configuration, then the software engineer should design the OFP software such that it will recognize when a message need not be transmitted. Then, when the message is not needed, the processor will not waste any additional processing time on that message.

Tools Design Issues

Excellent tools greatly increase the productivity of engineering teams. OFP debugging tools and automated tools for IV&V engineers are needed to decrease the time required to do software updates and to increase the accuracy and robustness of testing.

Among the first issues to be decided when developing tools for OFP debugging and IV&V automation is the choice of computing platform. The computing horsepower available in a common personal computer (PC) today is adequate for most IV&V tasks, and many companies sell circuit cards that upgrade PCs for many of the popular high-speed interfaces such as MIL-STD-1553B data busses. The PC hardware platform provides a very inexpensive (approximately $10,000,[4] compared to custom hardware designs which might cost $1,000,000 or more to develop) development environment that can be upgraded, and for which ongoing software enhancements are readily available.

Using a PC with 1553 interface capabilities, software engineering teams can write sophisticated and user-friendly tools that will extract data from the RWR via its 1553 interface. The PC can then store the OFP data for later viewing, or the PC can display the data in real time. These software tools are relatively easy to develop and maintain. In addition, many of the PC-based software algorithms that are developed for OFP debugging tools can also be used to develop automated IV&V tools. In particular, the low-level 1553 drivers and 1553 message parsing routines can easily be shared between OFP debugging tools and automated IV&V tools.

RWR Testing

At this point it is important to question why so much attention is being given to RWR testing. The fact is, it has been proven over and over that software such as an OFP is written once, but it is maintained forever.[5] While it is true that new software is written during every OFP block cycle update, the amount of time spent writing the software is dwarfed by the amount of time spent testing new software. Therefore, if the block cycle schedule is to be reduced, the testing process must be improved.

In a typical block cycle update, the software changes are identified, and OFP writers begin a cycle of modifying, testing, and retesting the software until the system appears to operate correctly. Then, the OFP is turned over to an IV&V engineer who will verify that the RWR meets the system requirements. If the IV&V engineer finds a deficiency, the OFP modifying and

retesting cycle is repeated. This cycle of OFP testing and IV&V continues until the system passes IV&V.

Because the IV&V process occurs near the end of the OFP block cycle update, the IV&V engineer is invariably going to be under severe scheduling pressure, through no fault of his own. At the point that an OFP is finally turned over for IV&V, the block cycle is usually well behind schedule, the first flight test is already scheduled, and everyone is panicked because no one wants to delay flight testing. Therefore, the IV&V process needs to be fast because schedules rarely allows a fair amount of time for IV&V. It should be as free from human error as possible, since engineers will be under a great deal of pressure and will be prone to mistakes. Finally, the process must be accurate because it is very embarrassing for an IV&V engineer to approve an OFP and then have multiple problems at a flight test. Automation of the IV&V process will help conquer all of these problems.

RWR IV&V Process

The traditional process used to do IV&V consists of three major phases: one-on-one tests, combinations, and dynamic scenarios. In the one-on-one tests, the test engineer generates signals to simulate every mode of every threat that the RWR is programmed to intercept. This is a "static" test, meaning that the simulation is done in such a way as to simulate a stationary aircraft against a stationary threat site. In the combinations portion of the testing, the test engineer picks selected combinations of threat signals that will stress the system; this is also a static test. In dynamic scenarios, the aircraft is simulated in motion against fixed threat sites on the ground. The following will focus on the one-on-one case, although the automation concept described will also work well for combinations.

Over the years, the one-on-one testing phase has grown from less than a hundred to nearly 500 "files" (a "file" is a threat generator script that generates an RF signal to emulate a particular threat in a particular mode). This one-on-one phase of IV&V takes two persons about one week to complete. During this one-week period, the IV&V engineer will run each threat file at only one selected angle and one selected power.

The traditional manual method for doing RWR IV&V works as follows. There is an IV&V engineer sitting in front of the RWR display with his list of threat generator files. There is a technician sitting in front of the threat generator computer terminal. The IV&V engineer calls out a filename. The technician enters computer commands to load and run a threat file. The engineer and the technician wait for the threat generator to start its simulation. The IV&V engineer observes the RWR display and marks a grade for that file depending on what he sees on the RWR display. This process is repeated for every threat and threat mode in the RWR MDB (500 or more files). It is really a credit to the dedication of the IV&V engineers and technicians that they can complete the one-on-one phase of IV&V in one week; it is a very laborious task.

The computing platform for the automated IV&V software can be a personal computer. The PC software can be written to follow test scripts generated by the IV&V engineer. From these scripts, the PC can direct the output of the threat generator, monitor the output of the RWR

under test, and grade the results based on pass/fail windows programmed by the IV&V engineer. Because the automated IV&V could be run outside regular work hours, it could be stated that it requires zero persons to run.

Automating the one-on-one phase of IV&V would accomplish the following additional positives objectives.

- ⇨ Using automated test methods, the one-on-one phase of IV&V can be completed in much less than a week. It is believed, but not proven, that an automated test method could complete the one-on-one portion of IV&V in one 24-hour period. Remember that the OFP editing, OFP debugging, and IV&V processes are a loop that is repeated until the system passes IV&V. Many times when an IV&V fails, the OFP writers make a "minor" software modification and resubmit the RWR system for IV&V. In these cases the RWR community is very tempted to skip complete IV&V in favor of a "mini" IV&V. If the one-on-one testing could be done in one day rather than one week, then this temptation would be much easier to resist.

- ⇨ Automated test scoring is quantitative, consistent, and objective. Humphrey states "Avoid nonreproducible or on-the-fly testing."[3] For the traditional IV&V process, the steps of the test are repeatable, but the scoring is not consistent because it is qualitative. Consistent scoring gives an indication of whether the RWR system is maturing over a long period of time and is constant regardless of the engineer assigned to lead the IV&V task.

- ⇨ Mathematicians would correctly argue that one sample taken in the manual IV&V process is not a statistically-valid number of samples. The automated process would allow for each threat file to be quickly tested many times to ensure that the RWR always gives the correct response.

- ⇨ Many more statistics can be measured. In the manual one-on-one process, the only measure of RWR performance that is measured is correct ID. Other measures of performance such as response time, detection range, DF accuracy, etc., are not measured. The automated IV&V could provide the data necessary to calculate these statistics.

- ⇨ Report charts can be generated automatically by the automated test software.

Summary

With the quantum improvement in semiconductor technology over the past ten years, RWR system designers have created RWRs with hundreds of thousands of bytes of nonvolatile memory. As inevitably happens when there is more memory, the operational software and databases have expanded to fill the newly available space. This expansion has complicated the task facing IV&V engineers.

In order for IV&V engineers to continue doing their work effectively, there must be improvements in the tools and processes available for accomplishing IV&V. Better tools must include better, quicker threat generators, but the most crucial element now missing in the IV&V suite is an automated method of conducting IV&V. With automated testing methods, IV&V measurements can be taken more quickly, and more thorough testing can be done. With a combination of improved threat generators and automated testing tools, it is possible to complete one-on-one testing within a single 24-hour period, including exhaustive tests of correct ID, DF accuracy, detection range, and response time for all threats and threat modes.

Acknowledgments

Many people at Georgia Tech Research Institute have contributed to this paper; thanks to all of you. Special thanks go to Lee Edwards, Jeff Murray, and Emily Warlick for their exceptional contributions.

Notes and References

1. D. Richardson, *An Illustrated Guide to the Techniques and Equipment of Electronic Warfare*, Arco Publishing Inc., 1985.

2. JED Staff, "A Sampling of Radar Warning Receivers," *Journal of Electronic Defense*, Vol. 15, No. 9, Sept. 1992, p. 97.

3. W. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Company, 1990.

4. A 50 MHZ 486 CPU with 16 MB RAM, 1 GB hard disk drive, 21" color monitor, and 1553 card is a suitable computing platform for automating the IV&V process. This PC can be purchased for less than $10,000.

5. D. Coleman et al., "Using Metrics to Evaluate Software System Maintainability," *Computer*, Vol. 27, No. 8, Aug.1994, pp. 44-49.

Streamlining Automated Test File Generation

Jerilee A. Rura
Andrea W. Ply
Science Applications International Corp.
Dayton, Ohio 45432

Abstract

Before the end of this century, the rapidly increasing number, size, and complexity of embedded Operational Flight Programs (OFPs) will present an enormous testing and support burden to the Air Force. *AutoVal*, the automated testing toolset pioneered by the Avionics Logistics Branch of Wright Laboratory (WL/AAAF) and SAIC, is a key technology for solving this problem. *AutoVal* has demonstrated more than a 100-to-1 reduction in the time required to perform OFP validation testing. While *AutoVal* has completely eliminated the test engineer hours needed to actually conduct an OFP validation test, test engineers still have to generate the test command files, initiate the test, and review the test results. Generation of the test file, the commands that orchestrate the operation of the test station to stimulate, monitor, and verify proper OFP operation, has now become the most time-consuming step in the testing process. SAIC is making two enhancements to AutoVal to streamline test file generation. We are making *AutoVal* more user friendly by adding a language sensitive editor and on-line help to it's X/Motif-based graphical user interface. We are also adding a "Learn Mode" that will speed the development of test files by monitoring a test engineer's actions while manually performing a test and automatically creating a test file to reproduce those actions. This paper describes these new features and how they will further reduce the cost and increase the effectiveness of testing embedded software in modern aircraft.

Background

Under the sponsorship of the Embedded Computer Resource Support Improvement Program (ESIP), WL/AAAF has conducted research to reduce the cost and improve the performance of avionics software support and testing. WL/AAAF investigated ways of reducing the time required for Operational Flight Program (OFP) validation testing. This research led to the creation and evolution of the Automated Validation (AutoVal) Tool Suite. Recent modifications to the AutoVal environment include an extensive redesign of the user interface, a command palette to assist when editing test scripts, and a learning feature used to generate command files. These new items are discussed in detail in the following sections. This section highlights AutoVal's main features and benefits. *Refer to the papers listed in the Bibliography for more background and a detailed review of AutoVal's capabilities.*

The AutoVal Program allows a user to perform real-time testing of emulated OFP software software by completely automating the actions of the test engineers. AutoVal automates the

testing process through direct control of the host test station. It performs both the stimulus required for the test and verification of the test results. The user manipulates the test station simulation using a test-oriented command language, which provides access to the system through shared memory and performs logical, arithmetic, and decision-logic operations. This command language resembles natural English and is carefully tailored to the testing environment. Test engineers can generate reusable command files that configure and initialize the simulation, control the simulation flight dynamics and avionics modes, verify test results, and automatically create test logs. AutoVal can be used to support all phases of testing from static unit test, through subsystem and integration test, to formal qualification test.

The AutoVal test environment may be tailored by the user via test command files, macros, switches, and symbol definitions to support a variety of different applications. At the highest level, test command files are created to express the validation and reporting process for the application. Macros are defined to tailor and expand the native AutoVal command language. Switches and symbols tailor the test environment and provide a familiar low-level interface to the application system.

Command files can be small units used to encapsulate a function or procedure on the application system. Conversely, command files can be large, complex test cases utilizing many building blocks in the form of other command files, macros, switches, and native commands. Command files change and evolve to match the testing requirements of the system being tested. Command files are the highest level tool, available within AutoVal, which can be used to tailor the test environment to a specific system under test.

Macros are user defined commands which can utilize several parameters. Macros can consist of generic, native commands, calls to command files, and other macros. For most purposes, once a macro is defined, it can be used over and over in the same way as any of the native commands. This allows the user to build libraries of commands which best fit the particular problem domain.

A large percentage of typical avionics and other test procedures involve actions performed by an operator (e.g., pilot) through various controls and control panels that influence the operation of the system under test. Many of the switches, buttons, and knobs used to input avionics data are very unique both in appearance and function. AutoVal needed a generic way to allow users to define not only the syntax of a switch (i.e., its name and positions), but also to associate the desired behavior with each switch activation. AutoVal's switch commands provide just that. Users define names, attributes, and position titles. Distinguishing between push-buttons, toggle-switches, and knobs is completely controllable by the user.

A feature used to tailor the AutoVal command language to specific applications and individual users are the symbol definitions. AutoVal symbols are used to provide a symbolic representation of system data. This allows test engineers to reference data using meaningful, symbolic names. In addition to making command files more useful and understandable, shared memory changes, minor system modifications, and recompiles don't require large changes to existing command files when symbolic representations are used in command file development.

X-Windows/Motif User Interface

A new AutoVal user interface (refer to Figure 1) has been designed and implemented to provide an efficient, integrated testing environment. The new AutoVal user interface is implemented using X-Windows/Motif and the Systems Engineering Research Center (SERC) Ada-Motif bindings. AutoVal currently runs on the Sun SPARCstation series of computers using the UNIX Operating System, specifically Sun O/S Version 4.1.3 with X-Windows(X11R4)/Motif(1.1.3).

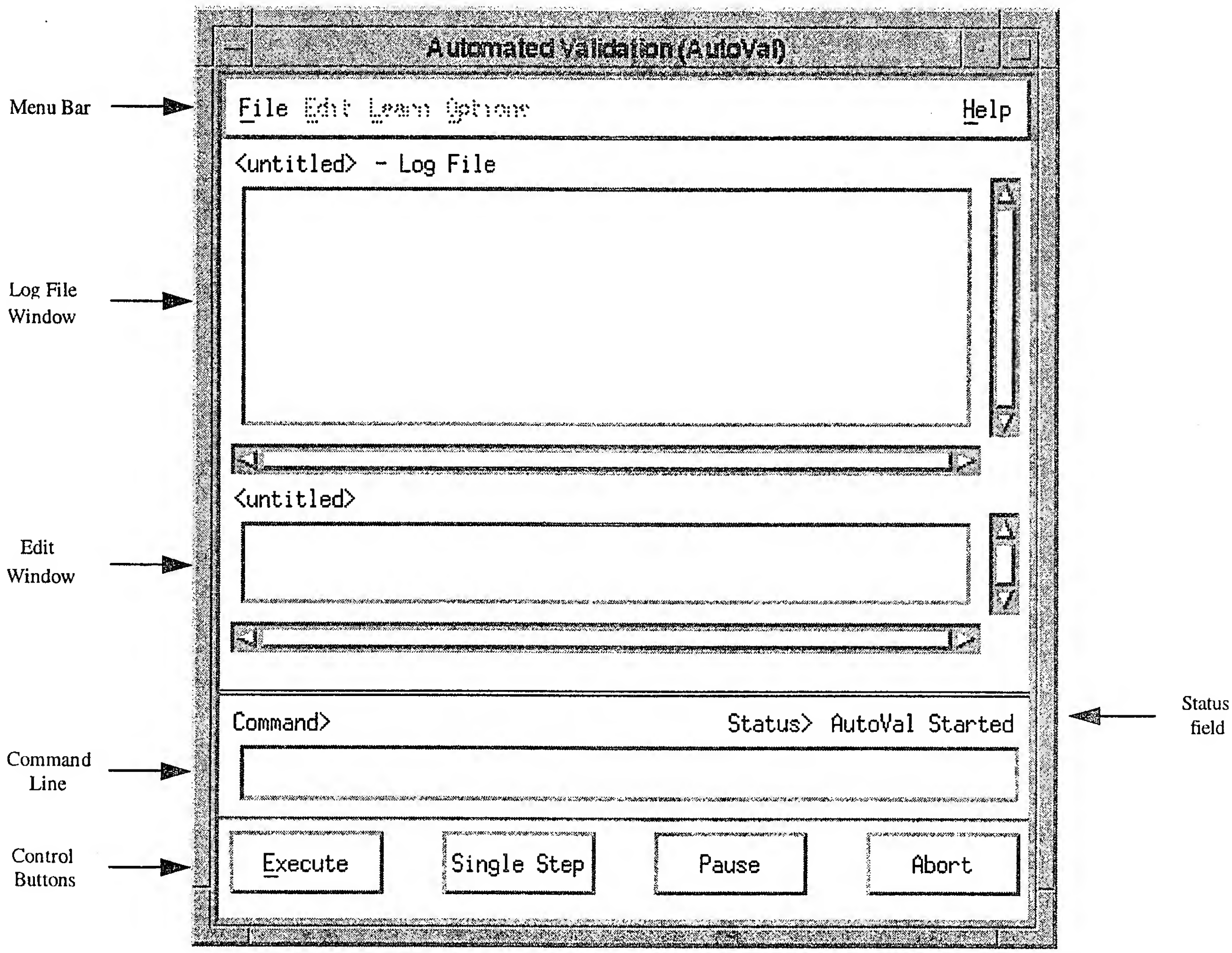


Figure 1. AutoVal Main Window

The new AutoVal user interface was designed to be easy to use. A tool with a cumbersome, complicated user interface is always a source of frustration for users. The user can move among the AutoVal menu options and windows easily using the mouse. Most menu options may also be selected directly by typing the corresponding "hot keys" displayed to the right of the menu selection. For those who prefer more hands-on control, AutoVal's unique command line feature incorporates an interactive capability to provide immediate feedback during testing and development.

The AutoVal Main Window, the central user interface component, utilizes a large form with a main menu for control of all primary AutoVal functions. This form provides a command line field for use during interactive command operation and incorporates an edit window for displaying and editing AutoVal command files. The form also contains a separate log window for displaying log file data and error messages. It presents status messages in a status message field. The inputs and displays associated with each of these fields have been tailored to the needs of the interactive AutoVal user. Users may choose to exercise AutoVal interactively when: 1) creating new command files; 2) defining macros, switches, and symbols; 3) debugging user defined command files; 4) debugging the simulation environment; and 5) operating the system in a "hands-off" mode.

AutoVal allows the user to interactively execute individual commands or command files. This is accomplished by using the Command Line. This command line interpreter provides for the parsing and syntax/error checking of the commands.

The edit window acts as a history for all of the commands entered via the Command Line. The user can scroll through the edit window to view previously entered commands or to copy and paste text into the Command Line. The edit window also provides a centralized area for general editing and command file generation. The contents of this window can be written out to a text file to save commands that have previously been entered. The saved commands can then be loaded into the edit window for re-use or editing purposes at a later time. AutoVal permits the command file located in the edit window to be opened or closed during an editing session. During such a session, AutoVal provides standard Cut, Copy, Paste, and Delete menu options for command file editing. In addition, AutoVal allows the user to syntax check or “load” the contents of the edit window.

Each time AutoVal is executed, a log of various messages and command responses is displayed in the AutoVal log window. This scrolling log provides a reviewable record of the AutoVal session including all status, informational, warning, and error messages generated during the session. The log can be very useful to a user executing command files or using AutoVal for interactive testing. In addition, the log can be saved to a disk file for later viewing. The saved file is an ASCII text file and can be printed or viewed using appropriate tools. This saved file is valuable for producing test reports.

AutoVal *status* command. However, AutoVal also uses this field to display information when The Status field normally displays data written to the screen by the execution of an loading or pausing a file.

The new AutoVal user interface also allows the user to open multiple “view-only” files. These “view-only” files appear in separate display windows and can be used for reference while working in the main AutoVal window.

Command Palette

The Command Palette (refer to Figure 2) is a tool provided to assist the user with language sensitive editing of AutoVal test scripts when generating command files. The Command Palette is available to the user through the Edit menu of the main AutoVal window, and is generated in a separate X-Windows/Motif "window". The window contents are available to the user during the entire editing process until closed by the user. The Command Palette displays AutoVal native commands as well as previously loaded AutoVal macro, switch and symbol names in separate areas. Selections from the Command Palette are copied directly into command files in the AutoVal edit window. The user can quickly select items from the command palette, thus reducing the amount of time needed to type the name, look up the syntax, or remember the syntax and correct usage of each item.

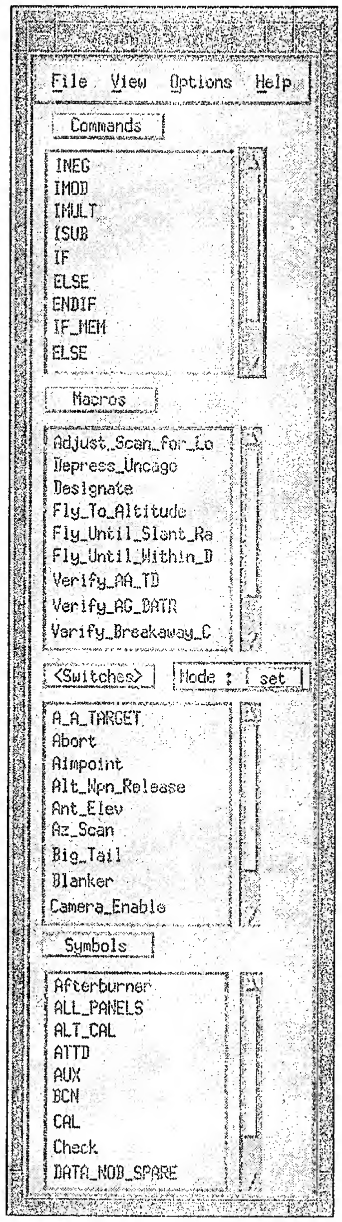


Figure 2. Command Palette

Many of the Command Palette design features were created and refined by building X-Windows/Motif prototype screens with UIM/Ada. UIM/Ada is a Graphical User Interface (GUI) builder for the Ada language. With UIM/Ada, X-Windows/Motif user interfaces are created quickly and easily. The tool was very intuitive to use and allowed developers to determine not only the overall look, but also the feasibility of a design. A more user-friendly Command Palette with easy to use and convenient features was the result.

The user interface component of the Command Palette consists of a form with a main pulldown menu and four scrollable display areas that list the names of previously loaded macros, switches, symbols, and commands. The menus allow the user to access the main Command Palette functions and features and often use cascade menus to display easy to access information. The help and exit options are similar to those used on the main AutoVal window. Using UIM/Ada, two prototypes of the Command Palette were designed. A narrow window displaying the four list areas in a single column, and another displaying two lists per column. As the Command Palette will be viewed on the same screen as the main AutoVal window and other windows, the narrow design was used to avoid a cluttered screen for the user.

When the user invokes the Command Palette, the initial screen alphabetically displays all commands, switches, symbols and macro names. This is potentially a large amount of information, particularly for symbol definitions. The user can choose to reduce the amount of data displayed in the macro, switch or symbol lists by simply selecting a filename. This filename acts as a filter on the appropriate list thereby only displaying data from that particular file. This results in a smaller, more concise list of data for the user to manipulate.

The user can access the list of filenames through the main menu or by a mouse press on the list title. The titles for the switch, symbol and macros list are designed to resemble push-buttons. Although these title labels do not operate as push-buttons, their appearance is a reminder to new users that these labels have an associated function. By using the mouse on this label, a popup menu displays the same list of filenames accessible via the main pulldown menu.

When a single file is selected for display, the notation, "< >", will be placed around the list title (i.e., < Switches >) to remind the user that a reduced list is being presented. This notation is used in lieu of displaying the actual filename, as filenames would be too long and cumbersome to display. Both areas allowing filename selection use radio button displays. When a filename is selected, the button for that filename appears depressed. The user need only click the mouse on the list title label to display the list of filenames and check the name of the presently displayed file.

A powerful feature of the Command Palette is the ability to supply syntax information for commands and macros. From the user's perspective, many of the commands and macros have complicated syntax which can be difficult to remember. For similar reasons, it is also desirable to provide a means to display the various switch positions. To implement these features, pop-up menus were designed for each individual list item (i.e., each command name). By using a simple mouse action, the user can easily select the name, or additional information if available, for commands, macros, and switches. (For symbols, only the name would be displayed.) Conceptually, providing popup menus for the user was straightforward. However, a simple Motif

list widget could not be used since it does not provide popup menus for each list item. As a result, each list item is designed as a separate widget (a text widget on a form in a scrolled window) to allow this functionality.

To use the Command Palette as an editing tool the user simply double clicks the mouse on a name in any of the lists or uses the mouse to slide down the popup menu and release the mouse over a selection. In either case, the selection is pasted into the AutoVal edit window.

Syntax using pre-defined switches in a command file always begins with the *set* or *turn* command. A convenient feature of the Command Palette automatically places either word at the beginning of a switch selection for the user. As *set* is the most frequent option, the initial Command Palette uses *set* as the default. The switch mode is displayed and can be changed in two locations. A main menu option, “Switch Mode”, provides the *set* and *turn* options in a radio box format. A push-button near the switch list title indicates the current setting. The user can toggle between the two options in the menu or use the push-button to make the change. When the push-button is pressed with the mouse, its label displays the other selection.

When the user invokes a popup menu for a display item, a “help” selection is included. Help for each item displays brief textual help to the user. The help information consists of text comments contained within the definition of the macro, symbol, or switch that describe the syntax and/or use of the particular item. The help for AutoVal commands provides more detailed information on their use.

Learn Mode

One recent addition to the AutoVal testing environment has been termed "Learn Mode" by WL/AAAF. Learn Mode takes the concept of automated testing to the next level -- automatic generation of AutoVal commands for OFP test procedures. When the system operates in Learn Mode, user actions are monitored and AutoVal commands duplicating these "test procedures" are generated. Test engineers can generate command files while performing day to day, familiar activities on the system, thus reducing the time to create a particular command file. The user doesn't have to remember AutoVal command syntax, macro names, or switch descriptions to duplicate their inputs.

In the current Learn Mode implementation, monitoring activities are performed by two, independent monitoring processes. Translation activities and Learn Mode state control functions are performed by a separate process (task) incorporated into the AutoVal main executable. The monitors, which are separate executables, communicate with the translator via shared memory. In this manner, the user can move and reconfigure the monitors as quickly and easily as other simulation software components. This allows the user to rearrange the simulation system if more processing time is required to monitor or translate data on a specific test station.

Learn Mode monitors three types of shared memory inputs: 1) Switches, 2) Aircraft Control, and 3) Simulation Control. Switches include such items as cockpit panel toggle switches, push buttons, and rotary knob settings. Learn Mode uses the AutoVal switch definitions to identify

shared memory locations for monitoring. Aircraft Control refers to autopilot inputs of altitude, roll, pitch, heading, etc. In the AutoVal and Learn Mode environments, aircraft control functions (e.g., pitch, roll, heading) are performed by using the autopilot panel and the real-time autopilot model. During Learn Mode, the user interacts with the normal simulation input devices and the aircraft autopilot to perform test procedures and validate results. Simulation Control refers to other user inputs to the simulation that are controllable via shared memory (e.g., test station configuration and startup). Learn Mode allows the user to identify these additional shared memory locations for monitoring.

When the user enters Learn Mode, the translator processes all switches currently defined in AutoVal. By default, the first *write* location and value encountered for each switch position are used as monitoring keys to determine data changes during Learn Mode. The AutoVal switch syntax was enhanced to permit the user to identify a "key" location and a mask value for a switch. If a key location is specified, Learn Mode scans the switch definition and associates each position with a unique value. This permits an AutoVal system engineer to insert simple switch keys for definitions which may be complicated or unusual. The mask value is applied if only part of the location is to be monitored and checked during translation. This feature enables the simulation designer to group more than one switch into a single shared memory location.

After extracting any pertinent information, the Learn Mode translator constructs a translation table to be used when recording. The table is indexed using the shared memory location and value setting for each valid switch position. For any valid location and value, the corresponding AutoVal switch command is stored and retrieved at the appropriate time during processing. User monitored actions, not corresponding to translation table entries, are duplicated by generating a corresponding AutoVal *write* command.

In addition to monitoring user defined switches, Learn Mode provides the user with a mechanism for identifying additional shared memory locations to monitor. The Learn Mode Selection Tool (refer to Figure 3) was specifically designed to give the user this capability. The Selection Tool allows the user to create, open, edit, and save groups of symbols for use during Learn Mode.

The symbol names are symbolic representations of shared memory locations within AutoVal. The selection tool has a main menu and an area for adding, deleting, or editing a list of symbol names. When the user has finished with modifications to the symbols, the whole list can be saved to a disk file or "save set". These save sets can be opened, modified, or renamed as needed.

For each symbol item in the saveset, the user can choose one of three options: Synchronous (real-time), Asynchronous(non-real-time), or Off. (The Off option signifies that the indicated symbol, though in the list, should not be monitored at all.) These three options are presented opposite each symbol name in a radio box for clarity of display and ease in modification. The Selection Tool provides consolidated display and editing capabilities for all extraneous Learn Mode monitoring locations.

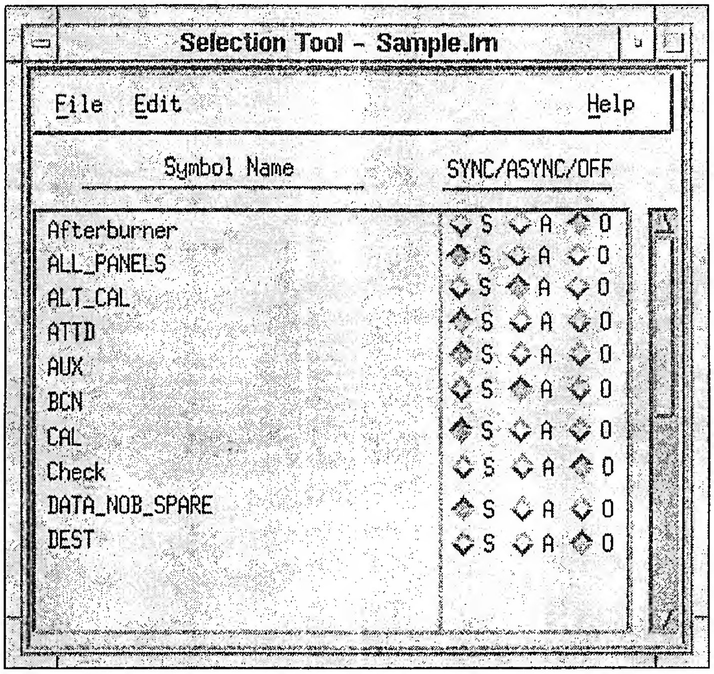


Figure 3. Selection Tool

Since simulation input device rates are usually different from one test station to another, Learn Mode needed to provide a real-time monitoring capability flexible enough to operate under such circumstances. To insure that user inputs are captured synchronously and quickly enough to preclude any data loss, Learn Mode utilizes a real-time simulation model to perform the majority of the monitoring activities. This real-time model conforms to specifications for Advanced Multi-Purpose Support Environment (AMPSE) and Virtual Test Station(VTS) components and is scheduled by the Distributed Ada Real-Time Executive (DARTE). As a model, monitoring activities are automatically synchronized with other simulation input and outputs. The frame rate of the monitor can be easily adjusted (via DARTE) to match the frame rate of the fastest input device to ensure that all user actions are captured.

During the design phase, it was noted that, in the AMPSE and VTS architectures, DARTE is not always active for all phases of test stand activity. This indicated a need for monitoring capabilities outside the real-time DARTE realm. A non-real-time monitor was incorporated into the design. To promote reuse and ensure that all monitoring activities and structures where consistent, both real and non-real-time monitors were designed to use the same Ada packages to perform input, computational, and output logic. Separate mainline logic for each monitor incorporates any differences required for each monitor type. The user can indicate specific data locations and switches to be monitored in a non-real-time mode. The non-real-time monitor is accessible during the entire Learn mode session while the real-time monitor is only active when DARTE and other simulation models can run. The non-real-time monitor is very useful for monitoring simulation setup and control functions.

Once the user has specified the valid locations to monitor and has entered the Learn Mode environment, the translator begins to communicate a variety of initialization information to each monitor including the number and actual shared memory locations that each process is responsible

for. The translator is responsible for distinguishing real-time vs. non-real-time data, partitioning the required data locations among the appropriate monitors, and creating the non-real-time monitor if needed.

Once this initialization phase is complete, the monitors begin scanning the system for user activity. When system values are updated, the appropriate monitoring process will identify the change and record the pertinent data. The translator and monitor processes use sections of shared memory as data buffers to pass this "change data" and error messages. The monitors write data and error indicators to the end of these circular buffers while the translator processes information from the beginning of the buffers. The monitors only deposit data into these circular buffers when an actual data change is detected, thus reducing message time and traffic. If processing is not sufficient to monitor all of the data in the required frame rate, the monitors indicate this error condition to the user via DARTE or the simulation display console. If high processing demands on the translator result in a backlog of data resulting in an overflow of the shared memory buffers, the user is notified that data has been lost.

When the translator receives change buffer data, it accesses the previously constructed switch translation table to identify the appropriate AutoVal command. As mentioned above, if no corresponding entry is found, an AutoVal write command is constructed to duplicate the activity. The translator then inserts the generated command into the edit window on the AutoVal main screen.

In addition to performing normal simulation activities in Learn Mode, the user may also interactively enter AutoVal commands at a command line prompt. This interactive capability enables the user to not only run macros, command files, and native AutoVal commands, but also to insert these commands into the edit window. Interactively, comments describing the test procedures, detailing problems, or highlighting edits can be inserted as well. Many of the features available during AutoVal Interactive Mode, such as syntax checking of commands, are applied before an interactive command is actually executed and placed in the edit window. This feature allows the user to switch between manually interacting with input devices (e.g., switch panels) and controlling them remotely by interactively entering commands and macros.

Upon exiting Learn Mode, the user may exercise any of AutoVal's editing or file management capabilities to manipulate, save, or empty the file. This capability allows the user to create small macros or large command files using Learn Mode. The user can also choose to insert new commands or procedures into a previously existing command file.

Summary

The AutoVal environment provides a flexible, tailorable approach to simulation software testing. Furthermore, recent user interface enhancements have focused on proving an integrated, easy to use interface to the AutoVal test environment. The AutoVal Command Palette is specifically targeted towards increasing efficiency when generating command files. Learn Mode takes an additional step in speeding up command file generation by automatically replicating a user's test station actions into a command file. These new features, when combined with the existing

capability and flexibility of the AutoVal software, provide a powerful testing tool which can solve a variety of software testing needs.

Bibliography

Steven A. Walters. "AutoVal (Automated Validation) Enhancements," Paper to be presented at the Test Facility Working Group Conference (TFWGCON) 95

Jeffrey Van Fleet, Steve Flannery, and Jerilee Rura. "Automated Operational Flight Program (OFP) and Simulation Validation - AutoVal," Paper presented at the National Aeronautic and Engineering Conference (NAECON) 94

Jeffrey Van Fleet and Jerilee Rura. "Automated Operational Flight Program (OFP) and Simulation Validation - AutoVal," Paper presented at the Test Facility Working Group Conference (TFWGCON) 93

Steven A. Walters. "Practical Techniques for Distributed Real-Time Simulation," Paper presented at the National Aeronautic and Engineering Conference (NAECON) 94

Steven A. Walters. "Virtual Test Station (VTS)," Paper presented at the Digital Avionics Systems Conference (DASC) OCT 93

BUILT-IN CHECKERS FOR REAL-TIME RADAR SOFTWARE

Carolyn B. Boettcher
D. Joel Mellema

Hughes Aircraft Company
PO Box 92426
Los Angeles, Ca. 90009
cboettcher1@msmail4.hac.com

ABSTRACT

Testing is one of the most critical and difficult activities during software development and maintenance. Nevertheless, only limited progress has been made in developing a theoretical basis for testing software. Proving that software will execute correctly for all possible inputs has been shown to be infeasible or prohibitively expensive for most applications. Ongoing University research in Program Checkers promises a dramatic breakthrough towards increasing the effectiveness of software testing by proving that a program produces a correct output for a particular input. However, until recently Program Checkers had been applied to only a limited class of non-realtime problems. A challenge was to see if this technique could be used in a significant way for testing real-time avionics applications. This paper reports on the efforts ongoing at Hughes in collaboration with Dr. Manuel Blum from the University of California at Berkeley, and the progress that has been made in developing and using Program Checkers for real-time radar software.

INTRODUCTION

Residual errors in mission-critical avionic software that result in faults or failures while the software is executing in the operational environment has been a continuing problem for the Air Force for which solutions have not been forthcoming. It has been shown that it is prohibitively expensive and may not even be feasible to test complex, mission-critical software to a very high degree of reliability [1], [2]. Program proving (i.e., proving that a program will produce correct outputs for all inputs) is even further from the state-of-the-

practice than exhaustive testing. Techniques commonly employed for life-critical systems with very high reliability and availability requirements such as employing redundant software and hardware to vote on the correct solution is also not considered a viable alternative for mission-critical systems because of the multiplicative increase in system cost.

As a method of significantly increasing the reliability of mission-critical avionic software without prohibitive increases in system cost, we consider a paradigm shift that changes the problem to be solved. Rather than proving a program operates correctly on all inputs, we instead prove that it operates correctly on a particular input. Then, rather than merely testing the software for some limited duration through which we try to convince ourselves it will always operate correctly, we employ a Program Checker that executes every time the program runs to verify its correct operation on the fly, both before and after it has been fielded. Under the new paradigm, we will, of course, continue to test software to some reasonable expectation of correctness, but there will also be the safeguard that faults will be detected in the event the software does not execute correctly for some input condition.

A Program Checker is an algorithm statistically independent from the software being checked that, given an input/output pair, decides if that input/output pair is correct. A complete Checker can increase the reliability of software by a multiplicative factor over conventional testing methods. Checkers can vary considerably in the amount of resources they consume and their probability of detecting any faults that occur. A range of Checker approaches are possible with varying costs and benefits. There are also a number of difficulties in applying Checkers in real-time avionics that must be overcome.

Program Checkers are intended to be permanently embedded in the software they check. They continue to check program results even while the software is operating in it's normal environment and could potentially be used as triggers for Smart Instrumentation (i.e., instrumentation that is triggered only when an anomalous event occurs). However,

if Checkers are embedded in operational flight programs, their impact on logistics and software maintenance must be considered.

The remainder of this paper discusses the background of the radar software testing problem, the theoretical basis of Checkers, and lessons learned from a limited experiment to transition Program Checker technology to working software engineers who are applying the technique on an actual radar operational flight program (OFP).

BACKGROUND

A substantial portion of the radar software life cycle cost is directly attributable to the effort required to test for and resolve errors in the most complex and mission critical components of the OFP. Testing of radar software goes through progressively more complex testing environments, starting with unit test performed in non-realtime on COTS platforms, proceeding to software and system integration facilities where testing is performed in real-time with actual embedded computers, and ending with flight test. These test environments provide increasingly more realistic scenarios in which to exercise the system. However, in progressing to more realistic test environments that are more likely to stress software and cause it to fail, it becomes increasingly more difficult to obtain the data needed to analyze and resolve performance problems. Program Checkers that are available in all the test and operational environments provide a method for signaling that a software fault has occurred and collecting just that data needed to resolve the error that caused the fault. Checkers may even be able to detect faults before they cause a system failure. Without a Checker, such a fault may not be detected every time it occurs, but be obscured until later when a system failure occurs, at which time the data needed to identify <u>where</u> in the software the fault occurred may no longer be available. Indeed, such a system failure may not occur until after the system has been deployed in the operational environment, at which time <u>no</u> internal software data is usually available.

In 1993 under Independent Research and Development, we began gathering information from software problem reports submitted to Hughes program offices to determine the software functions where test improvements are needed. The problem reports from a single radar tape upgrade delivery were mapped to software components and units to identify characteristics of the requirements, design, and/or code that led to undetected errors and reduced reliability of the system. The scope of reports extended from the time the software was delivered to the avionics integrator through flight test of the system. On the basis of that survey, we found that many residual problems remaining in the software after delivery result in externally detectable faults or failures only when a particular combination of events (for example, an unusual ownship and target geometry) occurs in the operational environment. Such problems are very difficult or impossible to detect with the scenario generation capability of the ground based radar integration laboratory. They are also unlikely to be detected during validation test using the current methodology because the anomalous behavior they cause is not always externally observable. As a result, we identified the need for a mechanism to detect software faults every time the faults occur even though anomalies are not always externally observable.

The Program Checker paradigm developed by Dr. Blum at the University of California at Berkeley shows promise of providing such a mechanism. As a result, in 1994, Hughes and the University of California (UC) jointly funded a project under UC's Microelectronics Innovation and Research Opportunities (MICRO) Program to see how this technology might be applied to computational problems that are important in real-time radar applications. Although jointly funded by industry and UC, effort under the MICRO Program is performed entirely by the University.

PROGRAM CHECKERS

The idea for a **simple checker** is derived from noticing that for some computations, the time required for the computation is an order of magnitude or more greater than the time

to calculate whether the result of the computation is correct. For example, in [3] the problem of computing a non-trivial divisor, d of a large integer, c is considered. It is much faster given (c,d) to determine if d is a divisor of c than to find d when one is merely given c. In [4], the definition of a **simple checker** is given as:

Let f be a function with smallest computation time $T(n)$. Then a **simple checker for f** is an algorithm (generally randomized) with the following input/output specification.

- **Input:** I/O pair (x,y),
- **Correct output:** If $y=f(x)$, ACCEPT; otherwise REJECT.
- **Reliability:** For all (x,y), on input (x,y) the checker must return correct output with probability (over internal randomization) $\geq p_c$ for p_c a constant close to 1.
- **"Little-*o* rule":** The checker is limited to time $o(T(n))$

Of particular interest to avionics, in [3] a Checker that resulted from the MICRO Program effort is presented for a Fourier Transform performed on the domain of fixed point fractions. Examples of simple checkers for other non-trivial numerical problems are also given in [5].

Unfortunately, there is at present no general theory on how to derive simple checkers for arbitrary computational problems. Moreover, although Checkers are required to take less time than the computation they are checking, if Checkers were used for every computation in a complex application like radar, the computational resources consumed would be unacceptably large. As a result, less computationally-intensive alternatives, although they might yield less benefits, are desired for real-time, embedded avionics. The challenge given to the University under the MICRO Program was two fold:

1. To extend the Checker paradigm to new classes of problems such as the Fourier Transform on fixed point fractions;
2. To identify techniques for checking computations that would be effective, yet have an acceptably low computational and memory overhead.

APPLYING CHECKERS TO A RADAR OFP

In response to the second challenge, also in [3], variations on the Checker paradigm are introduced Among these is the idea of a Partial Checker. A Partial Checker does not completely verify that an answer is correct; rather it checks only that some aspect of an answer is correct. During 1994, two teams of radar software engineers began an effort to see if the Program Checker paradigm, including less computationally intensive variations, can be applied to an actual radar OFP. The teams are working on two modes being incorporated into an update tape for an airborne radar: an air-to-ground Doppler Beam Sharpening [DBS] mode and a classified air-to-air mode. To determine the efficacy of the Checker paradigm for radar, the intent of the experiment is to embed Checkers into the OFP updates and record under what circumstances and how often the Checkers fire. Although the results of this experiments are as yet incomplete, we can report on some of the Checkers that have been incorporated and some that were considered, but proved to be too costly. The ongoing experiment is also limited by the fact that the radar mode efforts are not new developments, but are modifications of existing modes to add new features. As a result, a ground rule of the experiment is that Checkers are not added to code that is not otherwise being modified.

The following are examples of Partial Checkers that are being embedded in the OFP.

- Checksum of reordered averages with the unordered averages in signal processing code for zero range delay calibration.

$$\sum_{j=0}^{127} s(\text{ordered averages}_{\text{subj}}) = \sum_{j=0}^{127} s(\text{unordered averages}_{\text{subj}})$$

- Verify PRF stepping within the array does not deviate by more than specified amount in DBS data processing code for very high range and azimuth resolution

$$1 - \epsilon \leq \frac{|MAPRF(n) - MAPRF(n-1)|}{MAPRF(n)} \leq 1 + \epsilon$$ where n is the process sync index

- Verify the scaling factor does not exceed the limits, also in the DBS data processing code for very high range and azimuth resolution

  $$-MAXSF_{cde} \leq Sf_{cde} \leq MAXSF_{cde}$$

The following is an example of a Checker that was considered, but is not being used because of timing constraints.

- A check for loss of significance in the signal processing code for zero range delay calibration.

  $$\frac{1}{2} \sum_{j=0}^{16} \text{Raw Data}_{\text{subj}} \times \left(\frac{1}{8}\right) \equiv \sum_{j=0}^{16} \text{Raw Data}_{\text{subj}} \times \left(\frac{1}{16}\right)$$

CHECKER RECORDING

It is not enough to simply embed Checkers in the software. Some mechanism must be available for reporting when Checkers declare an error so that the error detection is externally detectable. The two teams of software engineers who developed the example Checkers described here defined a library routine that is called whenever a Checker declares an error. Information about the event is passed to the library routine through the calling sequence. The event information includes:

- Event ID (a unique error code number)
- Task ID
- Up to four words of data (expected values and measured values)
- Wall clock time

A circular buffer in memory is used as a temporary recording method. During flight test, information in the buffer will be permanently recorded on the instrumentation tape.

During integration test, the system can be halted whenever a Checker declares an error event.

Although at the present time there are no plans to make use of Checker events in the OFP after it is fielded, we have considered the feasibility of using the Built-in-Test matrix that is used to capture hardware faults for recording Checker-detected error events. However, the cost and logistics impact of that approach would have to be carefully considered. For the present, it is planned that, although Checkers might continue to execute in delivered systems, their firing would not be recorded or be in any way visible to the user during normal operation of the system.

FUTURE WORK

The experiment to embed Checkers in a radar OFP has not been completed. As the software with its embedded Checkers proceeds through testing and validation, we intend to collect statistics on whether and how often the embedded Checkers detect an error condition. We will also measure whether any Checkers declare an error when the software is in fact correct and will collect qualitative information about how well the Checkers aid the software problem detection and correction process. Finally, in addition to completing this limited experiment to demonstrate the value of Checkers, under the continuing MICRO Program, the University will investigate Checkers for algorithms such as the Kalman filter, PRF selection, and automatic target recognition.

CONCLUSION

Based on the limited results so far in extending Program Checking theory to algorithms used in real-time radar applications, we believe that this technology will prove useful in detecting difficult-to-find errors in complex avionics software that have proved resistant to current testing methods. We are excited by the prospect of the benefits that will be derived

from embedding partial Checkers in radar OFPs and collecting just-in-time information during integration and flight test whenever those Checkers detect an error event.

ACKNOWLEDGMENTS

We are deeply indebted to Manuel Blum and Hal Wasserman for their patience and dedication in teaching us about Program Checkers and their creativity in devising ways to apply this research to real-time embedded applications. We are also grateful to the University of California MICRO Program and Doc Dougherty, Technology Director at Hughes Radar and Communication Systems, for funding their efforts in extending the theory to practical radar applications. Finally, a special thanks to the two teams of practicing software system engineers at Hughes who worked with Manuel and Hal to implement Program Checkers in an actual radar OFP.

BIBLIOGRAPHY

[1] Hamlet, Dick, "Are We Testing for True Reliability?", IEEE Software, July, 1992, pp. 21-27.

[2] Butler, Ricky, Finelli, George, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software", IEEE Transactions on Software Engineering, Vol. 19, No. 1, January, 1993.

[3] Blum, Manuel, Wasserman, Hal, "Program Result-Checking: A Theory of Testing Meets a Test of Theory", Proc. 35th IEEE FOCS, 1994.

[4] Blum, Manuel, "Designing Programs to Check their Work", ICSI Technical Report TR-88-009, 1988.

[5] Blum, Manuel, Raghavan, Prabhakar, "Program Correctness: Can One Test For It?", Information Processing 89, G.X. Ritter (ed) Elsevier Science Publishered B.V. (North-Holland)

Automated Software Regression Testing

Gary K. Miyahara*
Rick K. Taketomo
Steve Tomashefsky

Hughes Aircraft Company
Radar and Communications Systems Segment
P.O. Box 92426
Los Angeles, CA 90009

Abstract

Automated Software Regression Testing is a means of reducing defects in operational flight program (OFP) software entering flight test by speedily detecting induced defects during system integration/flight qualification testing. This test method, in turn, allows test personnel to focus more time on finding new defects before flight test. The goals of this paper are to:

- Place automated software regression testing in the context of OFP software defects reduction.
- Describe options for implementing automated software regression test systems and the impact of these options on test facility design.
- Present the current status of plans on the F-15 APG-70 Process Improvement for Verification Testing (F-15 APG-70 PIVT) program[1] to install an automated software regression test capability on a manual-based testing facility.
- Describe how the F-15 APG-70 PIVT program will reduce defects in OFP software entering flight test by detecting defects early in the software development process.
- Recommend design considerations for builders of new test facilities.

Problem Description

Defects

The task of developing new software, as well as the task of modifying existing software, often injects defects into the software. This is particularly true of operational flight software, which operates in real-time among several processors. The reduction of the number of defects, as well

---

[1] The F-15 PIVT program is an on-going study sponsored by Warner-Robins Air Logistics Center F-15 Engineering.

as the minimization of the negative consequences of defects, is a significant part of an ongoing effort to continually improve the software development process. An examination of types of defects, and their impact on software development, follows.

Software defects can be categorized in either of two major classes: (1) new defects, which are related to the performance of new software and (2) induced defects, which cause existing software to behave incorrectly.

When new software is introduced, either as a new product or as a new portion of existing software, its purpose is to provide new capability or functionality to the software product. A new defect is an entity that causes the new software to be unable to correctly provide that additional capability or functionality.

The modification of existing software, as well as the introduction of new software, can cause induced defects. The existing software has previously passed some incremental testing, minimally at the Computer Software Unit (CSU) level, and possibly the Computer Software Component (CSC) level, or beyond.

The software development process is composed of the following nine steps:

1. requirements definition/analysis
2. design
3. code
4. CSU integration
5. CSC integration
6. Computer Software Configuration Item (CSCI) integration
7. system integration
8. flight qualification test
9. flight test

Although defects are introduced primarily during the first three steps or phases of the software development process, opportunities to detect induced defects occur in every phase of the process. The two phases where defects have the potential to dominate software development cost are defects in requirements definition/analysis and defects passed from system integration/flight qualification testing to flight test.

If a defect is made in requirements definition/analysis, the software effort may be headed down the wrong path, potentially wasting considerable effort. This class of defects is out of the scope of automated software regression testing.

If a defect is passed from system integration/flight qualification testing to flight test, the cost of detecting and analyzing the defect in flight test is very expensive because of the cost of using instrumented aircraft, flight testbeds, and a test range with data acquisition and data reduction

equipment. In addition, achievable radar defect analysis productivity is reduced by limited test access. Limited test access increases the dependence on highly experienced personnel. Usually tiger teams of the best and brightest (expensive) government and contractor talent are needed to fix problems at this stage. The cost of these experts is a significant portion of the cost of correcting defects found in flight test, not to mention the additional pressures of schedule, customer demands, etc.

Testing Methods and Tools

Current testing methods tend to be manual-based (i.e. non-automatic, labor-intensive) processes, especially for older systems like the APG-63 or the APG-70. An effective manual-based process has the following characteristics: repeatability, optimal number of steps, valid testing, and repeatable results. However, there are several problems with this process:

- Availability of test resources
  Because of downsizing in the Department of Defense, fewer government and contractor facilities are available resulting in a reduction in the number of hours that test beds are available for software testing. In addition, a combination of defense downsizing and the reduced numbers of appropriate personnel now entering the job market as a result of the baby buster generation, fewer workers are available in the youngest working classes to support test facilities.
- Skill level required for test personnel
  Because of the currently perceived lack of a long-term career in aerospace/defense-related businesses, highly skilled personnel may not be available to perform complex test facility tasks. The school systems are not capable of producing personnel with an adequate skill/knowledge level.
- Test validity
  A manual-based process relies heavily on the skill, knowledge, and proficiency of test personnel. Since these qualities vary from individual to individual, detailed procedures followed, interpretation of displays, and interpretation of results are subject to variation. This variation has a potential impact on the validity of tests.
- Test facility accessibility
  Labor-intensive testing generally requires significantly more set-up and operation time than automated testing, increasing the amount of time required to perform a given amount of testing. This increases the demand on test facilities, and reduces accessibility of test resources for others.

Test Facility Development Process

Sensor system test facilities are usually developed during each system Engineering and Manufacturing Development (E&MD) program, and the test facilities go into maintenance along with the sensor system. Major innovations in test facility design usually occur when the next E&MD phase occurs, when the government issues a technology contract to improve certain aspects of test facility functionality, or when sensor system maintenance transitions to an Air Logistics Center. Generally, there is no pre-planned product improvement program for the test facility as there is for the sensor system hardware configuration items or the sensor system software configuration items.

Historical Perspective

Most older test facilities started with manual-based test systems. Test facilities seem to have evolved in roughly the following manner:

- The first test facilities were based on hardware-based monitor panels and oscilloscopes, with a host computer to load the OFP into the sensor system under test, and stripcharts and analog instrumentation tapes to record results. Commercial off-the-shelf (COTS) computing systems were used as software development platforms but were not available to perform real-time functions.
- Later, logic analyzers with stand-alone capabilities (front panel control only) were developed and adopted. Standardized logic analyzers and instruments (IEEE 488 and VXI) with some computer control followed, but storage capacities were relatively limited. Instruments adopted disk operating system (DOS) personal computer (PC) hosts and command line interpreter interfaces. Command line interpreter interfaces were used for stimulus and data acquisition systems as well, with a wide variety of operating systems.
- Eventually, storage capacity increased and digital tape was introduced for instrumentation recording. Instruments adopted standard configurations, such as UNIX®-based platforms and Microsoft® DOS/Microsoft Windows™ platforms as hosts. Stimulus and data acquisition systems started standardizing on platforms such as UNIX, VXWorks®, and Microsoft Windows. Graphical user interfaces (GUIs) gained acceptance as well.

Newer test facilities have generally taken advantage of technology improvements. Older test facilities are usually a mixture of the equipment developed during the three phases of test facility development described above. Funding restrictions, cost/benefit concerns, test facility revalidation

concerns, test facility developer paradigms, test facility owner paradigms, and test facility user paradigms have all contributed to the lack of technology investment in existing test facilities.

Design Options

As stated earlier, the cost of fixing defects detected by testing is greatest for defects passed from system integration/flight qualification testing to flight test. Because automated software regression testing will detect induced defects earlier in the software development process, test personnel will be able to spend considerably more time on finding and debugging new defects.

In designing an automated test system to replace a manual test system, repeatability, optimizing the number of steps, test validity, and repeatable results are critical to success. The degree of automation for system-level testing depends on the anticipated return on investment. For this return on investment to be significant, the task or process to be automated must be repetitive and prone to human error (complex or long), and the software to be tested must have a high expectation of being retested, either by being a part of a software entity on a long term upgrade or maintenance path, or through reuse. In addition to these considerations, some manual control capability must always be retained in a test facility design to handle investigation and debugging tasks when a defect is found. There are several factors to consider in determining the degree to which testing should be automated. A description of the key elements in a test facility follows, which create a basis for the implementation of automated testing by the F-15 APG-70 PIVT program:

- Control (for both the tester facility and the sensor system controls)
    - Manually controlled (hardware switches)

      Several human factor problems emerge as the number of switches increases. Eventually, test personnel cannot reach all the switches from a single, convenient location and cannot toggle them effectively in real time.

      The decision to use actual sensor system controls instead of simulated controls must be driven by testing requirements, particularly when the sensor system controls (actuated by the operator on the deployed system) are part of another avionics system, such as the central/mission computer or the displays. The choice depends on whether the purpose of the test is weapon system integration or sensor system integration. For weapon system integration, actual controls should be used. Simulated controls could be used (because real controls are not available or too costly), but

then a judgment must be made on whether the weapons system software is being properly tested by the regression test (the real goal) or the simulators are being tested (a test of questionable value).

If the nature of the test is to support sensor system integration, simulated controls can be used, or the data bases and variables affected by the controls affect can be manipulated directly.

- Computer controlled with a command line interpreter user interface

  This approach requires test personnel to learn and remember the command language and syntax, a requirement that may become unreasonable as test facility complexity increases. For example, in a simplified radar test facility, the test facility subsystems include at least an avionics simulation, which includes navigation, a mission or central computer, and display simulations; a radar target stimulus simulation; and a radar processor test equipment function to load, dump, and control the radar processors.

- Computer controlled with a command line interpreter interface and scripting capabilities

  Scripting is very beneficial for test personnel because command files with embedded scripts can be used for repetitive tasks. However, command languages and syntaxes must be well understood to build new test cases.

- Computer controlled with a GUI

  This approach frees test personnel from needing to know a command language and syntax. The system is still manual in the sense that test personnel must manually use the GUI to perform the test steps for each test case, unless a method of recording and storing mouse positions, mouse clicks, and keyboard inputs is accommodated.

- Computer controlled with a GUI and scripting capabilities

  A scripting capability and a script editor for a GUI enable test personnel to edit a test case easily. The preferred implementation is to layer a GUI on top of an existing command line interpreter user interface and use the command line interpreter commands and syntax for scripts. Experienced users of the command line interpreter interface do not need to relearn a new command language to interpret the scripts.

  Some emerging COTS scripting systems may minimize the cost of developing a scripting system tailored for a particular facility.

- Simulation
  - Off-line, non-real-time simulation of sensor system inputs and simulated real-time inputs to an emulation of the system under test

    This approach is appropriate for software integration and system integration testing, especially when there is little or no feedback from sensor or software outputs to sensor inputs and little risk of timeline overrun.

  - Off-line, non-real-time simulation of sensor system inputs and real-time inputs to the system under test

    This approach is appropriate for software integration and system integration testing, especially when there is little or no feedback from sensor or software outputs to sensor inputs, but there is some risk of timeline overrun.

  - Off-line, non-real-time simulation of sensor system inputs and simulated real-time inputs and responses to an emulated system under test

    This approach is appropriate for software integration and system integration testing, especially when there is feedback from sensor or software outputs to sensor inputs and some risk of timeline overrun.

  - On-line, real-time simulation of the inputs and real-time interaction with the outputs of the sensor system under test

    This approach is appropriate for software integration and system integration testing, especially when there is feedback from sensor or software outputs to sensor inputs, but there is great risk of timeline overrun. Because these simulations are truly real time, these simulations are the most difficult to implement.

- Data Acquisition
  - Manual and visual acquisition of sensor system and test facility (simulation) data through the use of terminals, monitor panels, or test equipment displays

    These methods of data acquisition are subjective and very inaccurate. A flash on a display may escape a tester if he is looking somewhere else to perform a test equipment or sensor system control function. Additionally, it may not be possible for test personnel to determine the exact position of a symbol on a display.

◊ Computer-based acquisition of sensor system and test facility (simulation) data

Alphanumeric data (representing simulation parameters or sensor system variables) is easily handled by a computer-based data acquisition system. Display data is usually more difficult. Often, the sensor system does not actually control the display bit map and raster systems. Instead, it sends sensor data to a central computer or mission computer to be formatted for display. One method is to capture and compare pixel data with a known expected symbol pixel-by-pixel to determine whether the test passed. The other method is to capture and compare underlying sensor system variables and data bases with expected values before the data is converted and formatted for the display device, central computer, or mission computer.

The choice between these two display testing designs is driven by considerations similar to those discussed for the control options. A decision must be made about whether the nature of the test is to support weapon system integration or sensor system integration. If the nature of the test is to support weapon system integration, pixel-by-pixel comparison may be appropriate. However, if the central/mission computer or display device must be simulated, a judgment must be made on whether the weapons system software is being tested or the simulators are being tested.

If the nature of the test is to support sensor system integration, it is preferable to compare underlying sensor system variables and data bases with expected values because central/mission computer and display OFPs typically are developed in parallel with sensor OFPs. Updated non-sensor OFPs are usually not available during sensor OFP integration.

- Data Reduction (usually applies only to instrumentation data tapes)

◊ Manual setup by the operator as soon as instrumentation tape is available

◊ Automatic pre-defined reduction as soon as instrumentation tape is available

The purpose of a data reduction function is to handle flight instrumentation data tapes, aid validation of flight instrumentation software before flight test, and reduce test facility instrumentation tapes in cases where the bandwidth of the instrumentation data is too high to be handled by available computing systems. To support flight instrumentation data tapes, some capability to handle manual setup by an operator will be required. Automatic pre-defined

reduction may not be cost-effective unless the test facility has a multiple-shift operation.

- Data Analysis
  - Manually set up or interactively used as soon as instrumentation and facility test (simulation and lab instrument) data is available
  - Automatic pre-defined analysis
  - Automatic declaration of pass/fail as soon as data analysis is available

  Some manual capability must be preserved to support investigative and debugging tasks when defects are found. Automatic pre-defined analysis and declaration of pass/fail represents a large software investment and is appropriate for mature sensor systems in which system behavior is well characterized; that is, a fairly large amount of software is reused from one OFP build to the next.

- Test Case Management
  - Manual
  - Computer data base repository of test cases, test case coverage, and test case results
  - Computer data base repository of test cases, test case coverage, and test case results in which test case results are inputs from the data acquisition system and the data analysis system

  This approach also represents a large software investment and again is appropriate for mature sensor systems in which system behavior is well characterized. COTS data base management tools reduce the development cost of these systems, but the main cost is expended in maintaining the data bases.

F-15 APG-70 Process Improvement For Verification Testing (F-15 APG-70 PIVT) Program Status

The Warner-Robins Air Logistics Center Software Development Facility (WRSDF) program will culminate in the delivery of a manual-based testing facility to support the F-15 APG-70 OFP. Concerns about using a manual testing process for maintaining a complex OFP led to an Air Force-sponsored study and development effort called the F-15 Process Improvement for Verification Testing (F-15 PIVT) program to consider an automated OFP regression testing process for the F-15 APG-70 radar.

The WRSDF is composed of four configuration items: the Central Development Facility (CDF), the APG-70 Radar Test Bench System

(70RTBS), the Advanced Software Bench (ASB), and the Instrumentation Data Reduction & Analysis System (IDRAS). Figure 1 illustrates the support systems and their associated functions. The CDF consists of a mainframe computer with terminals that hosts the OFP software engineering environment. The ASB and 70RTBS are GUI-controlled radar test stations. The IDRAS has a mixture of command-line-interpreter-controlled and GUI-controlled functions. The IDRAS contains flight instrumentation data recorders that are shared by the ASB and 70RTBS. The facility has no automated software regression testing capabilities resident in the original design.

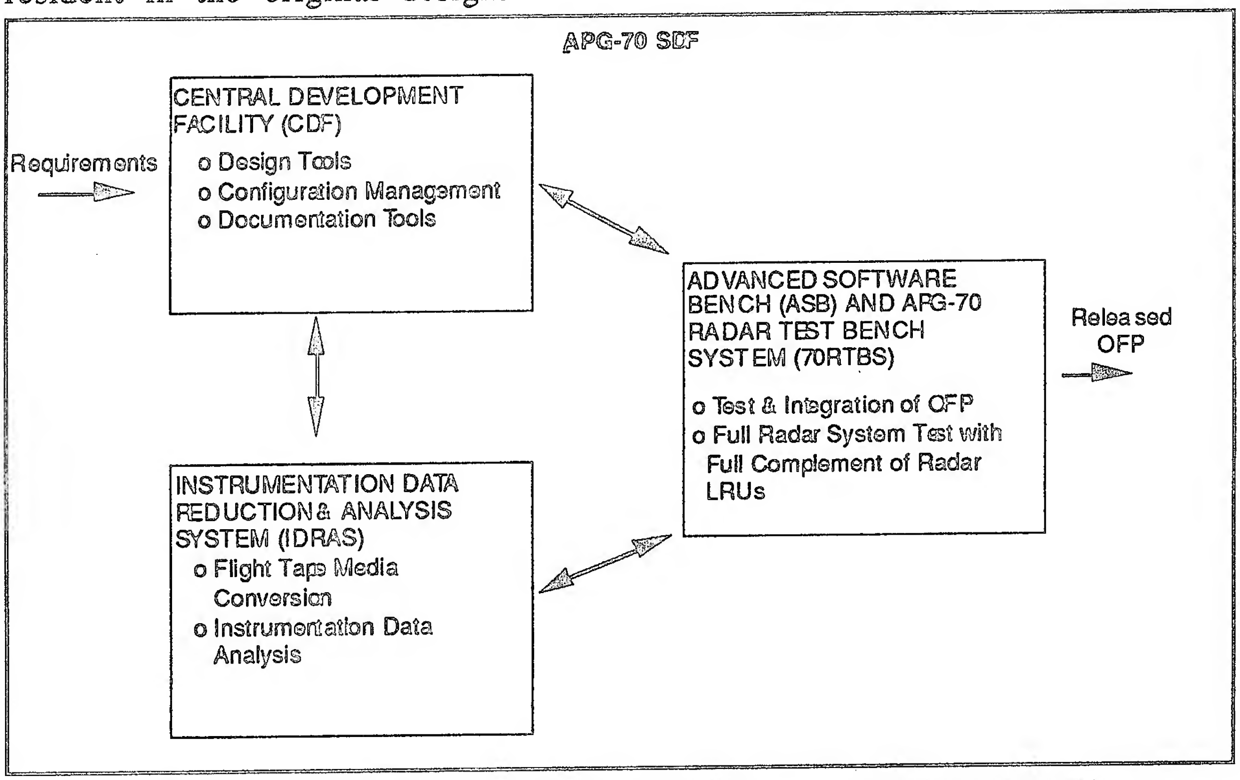


Figure 1. WR-ALC APG-70 Software Development Facility (WRSDF)

The goal for the F-15 APG-70 PIVT program is to maximize effective utilization of the WRSDF. Effective utilization of the WRSDF is critical to successful organic capability for WR-ALC in F-15 APG-70 software maintenance, which will transition from Hughes Aircraft Company to WR-ALC circa 1996. Effective utilization drove three design requirements:

- Reduce the requirement for highly trained specialists
- Reduce OFP Block Cycle release time and increase test content
- Reduce flight test hours and eliminate defects that get into flight test

The most significant element of software development in terms of cost, time, and process is test. More productive testing means more productive OFP development. Thus, the PIVT program has focused on improving testing productivity through a structured test approach (complete test coverage, minimum duplication, test planning/scheduling enablers) and test automation (i.e., automated software testing) to reduce variability caused by operator error. Faster test times, which allow more tests to be performed more frequently, are thought to be an added benefit.

Further analysis was made on the structure of the current manual method of performing regression testing on APG-70 radar OFP software. This method of testing concentrates on system-level regression testing. CSU-, CSC-, and CSCI-level regression testing tends to vary with the experience and knowledge of the test personnel. Test coverage was found to be complete, with little duplication. Automating the current manual system-level regression test was accepted as the baseline for automated software regression testing.

Since system integration testing is performed on the WRSDF 70RTBS and ASB and since only the WRSDF 70RTBS and ASB are designed with GUI-controlled capabilities, the PIVT program concentrated on automating the process of performing a test (stimulating the radar system and observing behavior) on the 70RTBS and the ASB rather than on using the IDRAS for initial automated software regression test prototyping. These WRSDF GUIs will serve as the basic source of control for the APG-70 PIVT-enhanced system. The primitive scripting capability of these GUIs will require improvements to control all 70RTBS and ASB functions through scripts. Script editors for the GUI scripts will need to be developed. The GUIs will require changes to accommodate simulated radar controls, which cannot be controlled effectively with the GUIs.

The WRSDF simulation systems require no changes other than the radar control simulations since the simulation systems were designed to be real-time interactive with the radar system inputs and outputs. The data acquisition systems will require major changes since the systems were designed to be used with an interactive user. Data is logged on several federated systems, but the data is not collected and correlated.

Current program status is that major design decisions are still required. A decision to determine the method of evaluating display data (between pixel-based comparisons and evaluating underlying variables and data bases) has not been made. Test data routing and data archiving evaluations are occurring. Also in progress is the determination of whether pass/fail decisions should be made in real time.

Recommendations and Observations

- As the OFP software is modified or enhanced, regression testing of the OFP is needed. Regression testing of the OFP software must be utilized several times over the life of the software to make a commitment to build automated software regression systems.
- Test facility software providing automated test capability should be designed to support various OFP versions. Maximization of automated test software reuse is important to maximization of the return on investment of implementing automated test.
- Test facilities require manual-based control capabilities to handle investigative and debugging tasks when defects are found. Upon the discovery of a defect, the test personnel should have the capability of interrupting the automated operation, utilizing test tools interactively to determine the nature of the defect.
- Test facility controls should be GUI-based with a scripting capability and a scripting editor. This allows the automated test facility to invoke test functions on a level of control similar to the level of control offered to manual test personnel.
- Sensor system controls may be either actual or simulated depending on whether the test is at the weapon system level or the sensor system level. The breadth of testing dictates the dependence on actual sensor system controls.
- Simulations and stimulus of the sensor system OFP may be in real time depending on the amount of feedback from sensor output to sensor input and the degree of timeline overrun risk.
- Evaluation of sensor system display information may be either pixel-based or sensor-system-data-based depending on whether the test is at the weapon system level or the sensor system level.
- Commit should be made for pre-planned product improvement programs for test facilities to ensure long-term support for weapon system development. As weapon systems change and grow, test facilities can not remain constant and be expected to provide consistant levels of support.

Glossary

ASB. Advanced Software Bench.

CDF. Central Development Facility.

COTS. Commercial off-the-shelf.

THREAT GENERATOR SUB-WORKING GROUP (TGSWG)

Co-Chairs:
Reggie Smith, Jerome Smith and Jan Breeden

RADAR ENVIRONMENT SIMULATOR
ENGINEERING TOOL, TEST SET OR TRAINER?

Edward J. Ayral

Cross Systems Division
AEL Industries
Alpharetta, Georgia 30201-7700

The admonition to "Simulate Before and During - Building, Testing, Buying and Fighting" is certainly *a-propos* to Radar Environment Simulators (RES). Long used in the laboratory as a design tool for the design of new radars and for the evaluation of hardware and software changes to existing systems, RES evolved to test equipment for first article testing and production acceptance testing. Now, because of the high cost of aircraft flight time and the danger of broadcasting sensitive EW information, RES is emerging as an embedded field training and performance monitoring system.

RESs have been defined as systems that stimulate a host radar at the RF, IF or digital level with simulated radar returns. The end use of the RES, however, necessarily determines its ultimate configuration. Rarely would a high fidelity system designed as an engineering design tool meet the cost constraints and scenario requirements of a trainer. As designers of RES systems, Cross is often confronted by potential users that desire to use a single configuration for all applications. This paper attempts to illustrate why a common design approach is not practical.

RES AS A DESIGN TOOL

Many forces drove the radar design engineer toward a laboratory system that could provide **REPEATABLE** electromagnetic environments for evaluating radar performance. Even disregarding the expense, many environments such as flocks of birds and swarms of insects can never be adequately tested in the field. Other requirements like hundreds of targets, jammers, and clutter, are impractical and time consuming when attempted in the field. FCC restraints on transmissions and other limitations have led to dependence on laboratory RESs for the evaluation of radar designs, hardware and software design changes, development of new ECCM to counter new ECM, etc.

When a radar designer specifies the requirements for a RES, he is concerned primarily with the following:

- High fidelity. In this case, as repeatable as possible and as near as possible to actual conditions. No surprises when the radar is tested in the field!

- Flexibility. Able to present different and new electromagnetic environments through reprogramming.

- Performance. The capability to stress the radars capacity to the utmost. Many targets, many jammers, severe clutter, etc.

- R&M. To a lesser degree and specifically not to hold up the program.

Since in most cases the designer is altering the radar to determine the effect of the alteration on radar performance in a repeatable environment, he is little concerned with an analytic capability that purports to identify a failure cause. The radar designer is also usually not concerned with size or environment because, as the name implies, the laboratory RES is usually permanently emplaced and used in the laboratory.

The configuration of the laboratory RES may also be somewhat dependent on the sophistication of the radar customer. If the customer specifies that a RES be designed and developed prior to or in consort with the radar design to be used for evaluation of the radar during development, and if air defense personnel are involved with the RES design, the end result may be considerably different than if the RES design is solely the responsibility of the radar designer.

FIG. 1 represents a typical RES used as a design tool.

RES AS A TESTER

In many cases the design tool RES suffices as the first article test set. First Article testing represents the ultimate determination that the design meets the criteria and is generally performed jointly by design and test personnel.

This is not the case for acceptance testing. When a program progresses to acceptance testing of production systems it is presumed that the design has been proven adequate. The acceptance tester, generally designed by the radar producer, must determine that each production system has been properly built. A simple test set less expensive than a RES can be built which is more effective and is concerned with the following:

- Pass or Fail. Each acceptance test must be defined in terms of absolutes which determine correct or incorrect manufacture.

- Analysis. When a failure occurs, the tester should be able to analyze the cause of failure and the corrective action required.

- R&M. More important than in the laboratory.

Often the test set is designed to be more like BIT than RES. Design emphasis is on analysis of failure cause and determination of corrective actions rather than on end to end performance.

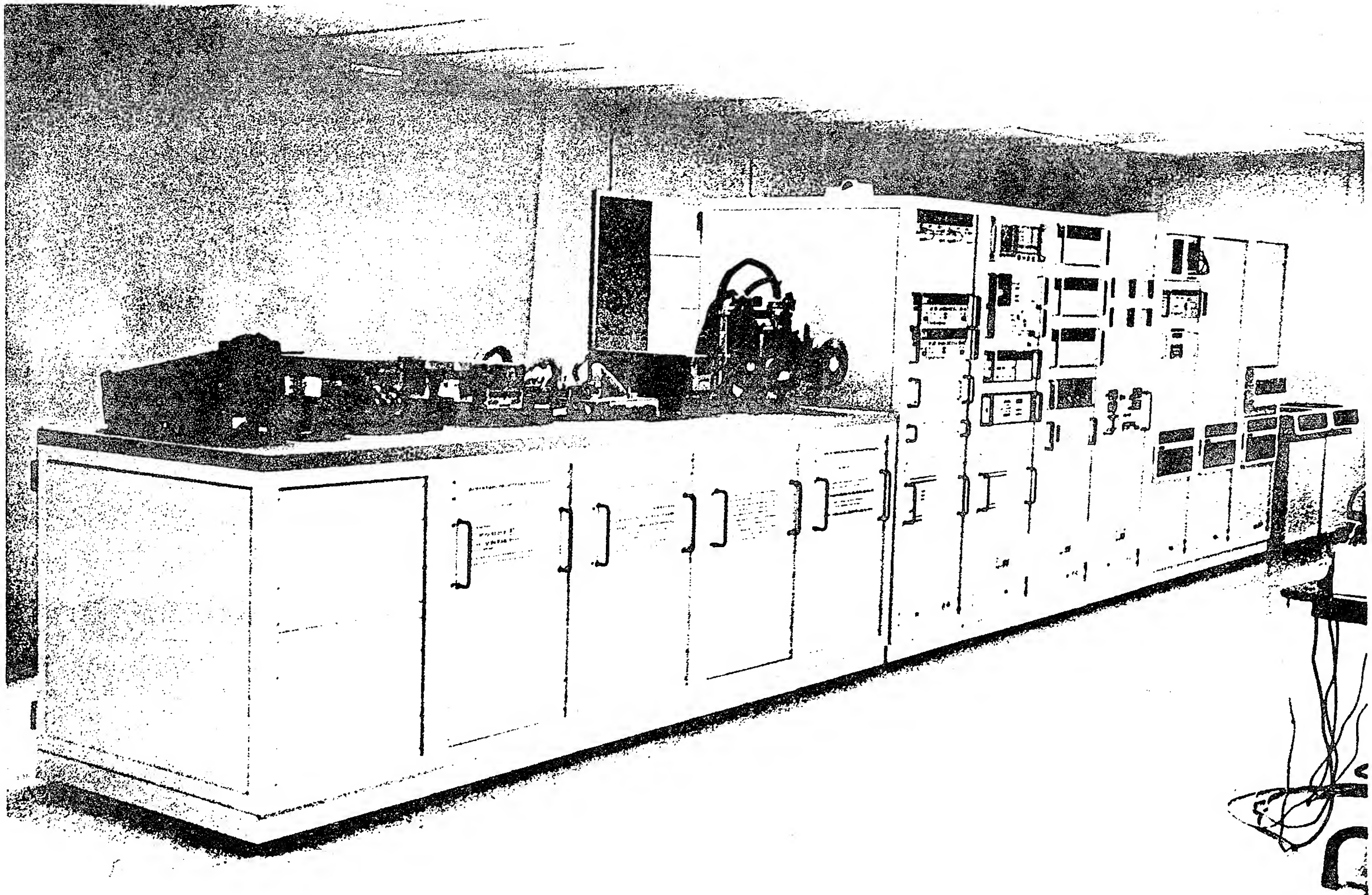


FIGURE 1

Typical RES Used as a Design Tool

RES FOR TRAINING

The greatest disparity between radar designers and air defence commanders is in the realm of training. Training to a radar designer encompasses how to use the operator switches on the radar. A radar designer's concept for a training system typically consists of the radar operating console and a PPI with a video display.

Conversely, an air defence commander would like to train with many real aircraft using jammers and ECM, and firing missiles. He would like to direct real interceptors and fire real SAM's. Defence commanders equate the radar designers concept of training to learning to drive by reading the owner's manual! The RES concept which injects RF into the front of the radar is a compromise that allows the commander to train using a large part of the defence system without radiating information and without the need of air support.

NATO first recognized the need for an embedded RF training system in the early seventies as the NATO Air Defense Ground Equipment (NADGE) was deployed. NADGE required that air defense radars be permanently placed near threatened borders. In this situation, training maneuvers with a large contingent of aircraft with jammers and other ECM equipment was not only expensive but dangerous. To counter this situation and to enable operations personnel on site and system wide to train in a realistic EW environment and thereby improve their mission effectiveness, NATO produced document "Annex II to Enclosure to DS/ADSD (78) 65" which required that a RES be supplied with each air defense radar. A later document stipulated that each portable radar be supplied with a plug so that a RES could be utilized by the radar commander whenever desired.

During the late 1970's the U.S. Navy recognized a unique training and skill retention problem within the Combat Information Center (CIC) aboard ship. CIC officers and radar operators were thoroughly trained in such skills as target detection and tracking, Identification, Friend or Foe (IFF) and jamming and chaff avoidance while in school using simulators and other aids. In addition, modern air defense radars are equipped with moving target indicators, auto tracking, and anti-jamming capabilities. When this combination of recently trained personnel and modern radar first put out to sea, the skill level of our sailors was formidable. But, while at sea or in port on duty, the opportunity to practice these skills proved to be extremely limited. In many ports, radars are not permitted to transmit. While at sea, except for planned exercises involving aircraft and other facilities far outside the ship's captain's authority, tracking and other skills could only be practiced when an occasional transport or other aircraft flew over. In this scenario, jam avoidance and IFF capabilities decayed while at the same time our adversaries radar avoidance capabilities and techniques were changing and improving.

To address both the loss of skill in the CIC and the incorporation of newly learned enemy techniques, the Navy equipped several aircraft with enemy mimicking jammers and chaff with the intention of scheduling "Fly-over" training missions. During the missions that were flown, inevitably, the aircraft were able to avoid radar detection and tracking. Once the Navy realized the importance of continuous training in this area, the logistics and cost of using altered aircraft

over a widely dispersed fleet of ships was observed to be economically unfeasible and the shipboard embedded RES concept was promoted.

Radar operators need to be trained in ECM recognition and avoidance. They need to understand under what clutter conditions they can reliably detect and track targets. Weapon controllers must often make split second life threatening decisions when confronted with many targets while being confused by ECM. Therefore, the training RES designer must be concerned with a large quantity of targets in real time as well as the capability of the radar to detect targets during ECM and clutter simulations.

Since the training RES is either embedded with the radar or delivered to a fielded radar for training exercises, size, weight and environmental considerations are important.

The training RES design must also be concerned with cost since the training RES is generally embedded with the radar on a one per basis. Fidelity is subservient to positive training. For example, it is important that ECM and clutter affect the radar the way real ECM and clutter will, so that the operator becomes familiar with expected attack scenarios, but no measurements of these effects need be taken. As many targets as can reasonably be expected in an attack must be presented and modified in real time. The training RES may generate signals digitally and inject at the digital, IF or RF level as a balance between cost and fidelity. It must be flexible enough to present different scenarios and ECM as Intelligence provides expected enemy capability. Since the training RES is often embedded with the radar, limited subjective testing is often required for use as a radar maintenance tool.

A typical embedded training RES is shown in FIG. 2.

FIG. 3 Represents a training RES for mobile radars.

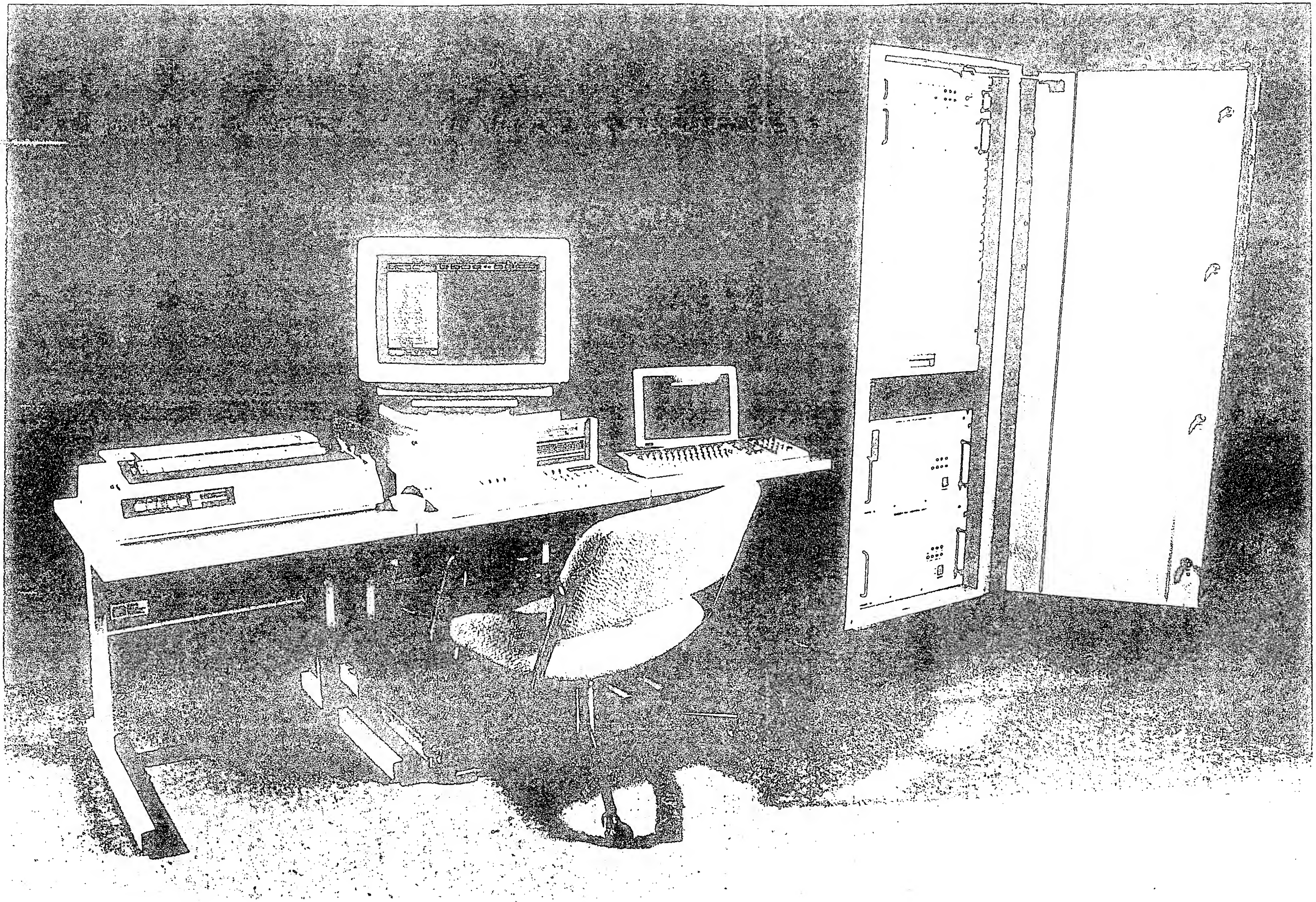


FIGURE 2
Embedded Training RES

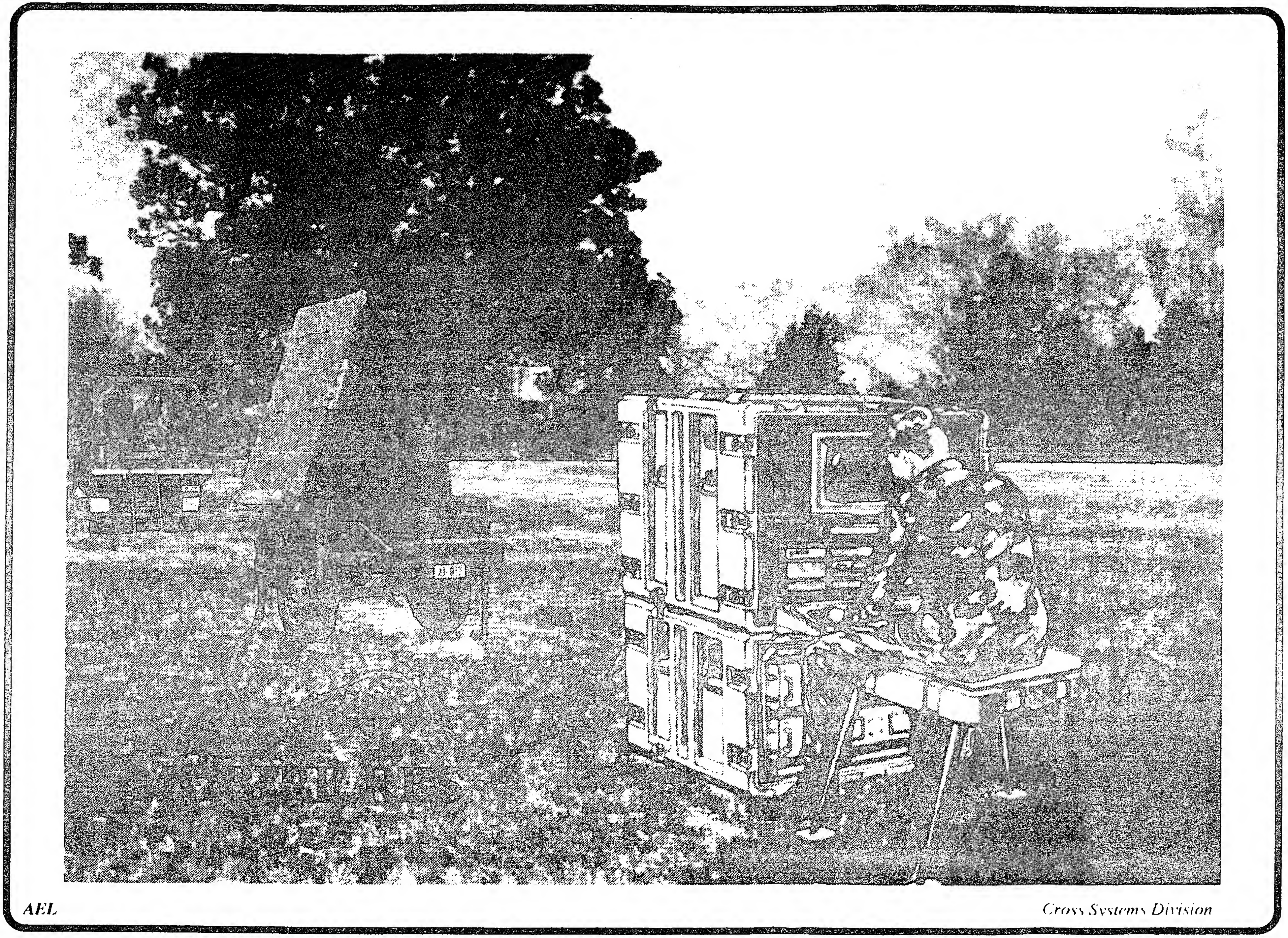


FIGURE 3
Training RES for Mobile Radars

The following chart summarizes the requirements for the various RES:

REQUIREMENT	DESIGN TOOL	ACCEPTANCE TESTER	EMBEDDED TRAINER
FIDELITY	High as possible	Suitable for measurement	Must affect the radar as the radar environment
REPEATABILITY	Essential	Essential	Limited importance
SATURATION-QTY OF TARGETS, JAMMERS, ETC.	Important	Not Important- By analysis	Essential for stress training
ANALYSIS CAPABILITY	Limited Importance	Essential	Not required
SIZE & WEIGHT	Not Important	Not Important	Critical
ENVIRONMENT	Laboratory	Factory	Field Transportable
COST	Relative to field testing	Relative to test time and fault analysis	To be produced in the same qty. as the radar

CONCLUSION

While it may appear that a common design will suffice for various RES applications, careful consideration of the requirements indicates that there are cost and other advantages to tailoring the design to match the specific criteria.

(1)

Theater Optical and Radar Code: A Unified Approach for Optical and Radar Threat Signature Generation

(2)
David Blair*
Stephen Clanton
Loren Dickerson
Jill Gothart
Kenneth Green
Cliff Liles
Anna Norton
Mark Paris
Garth Powell
Jody Smith
Randy Wood
Robert Oravits
(3)
SY Technology, Huntsville, Alabama 35816*
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SY Technology, Huntsville, Alabama 35816
SFAE-PEO-MD-TSD-TS, Huntsville, Alabama 35816

Abstract

SY Technology is developing an integrated optical and radar target signature code for theater missile defense. The Theater Optical and Radar (ThOR) code is designed to model threat targets from launch through impact (or intercept), including aerothermal effects, environmental heating, thermal response, optical/radar signature, and sensor effects. The goal of this program is development of a single, high-fidelity signature generation capability for use by interceptor programs. Current uses include analysis of flight test data, characterization of threat systems, evaluation of the effects of countermeasures, and evaluation of system performance in realistic scenarios. The modular nature of ThOR allows the integration of existing, industry-standard codes such as TSAP for trajectory calculations, a modified LANMIN for aerothermal calculations, and X-Patch for static radar cross section calculations. ThOR is designed to allow substitution of other codes if desired. Although there are existing separate codes which can calculate aerothermal effects, optical signatures or radar signatures, there has been no single code which allows consistent, realistic target modeling for all of these applications. ThOR provides a graphical user interface (GUI) through which the user defines all input information, including trajectory and environment, target dimensions and materials, and sensor parameters. Sensor parameters include not only location, but also specific sensor properties such as spectral response function, focal plane array type, quantum efficiency, detector responsivity, and aero-optic effects. ThOR is unique in that it ensures that both radar and optical signatures are based on the same target model. ThOR uses a knowledge-based GUI to evaluate input for consistency and to screen errors before code execution. ThOR includes a graphical data visualization application called Advisor (Advanced Visualization of Simulation or Reality), which allows the user to analyze both simulation output and actual measurement data. Advisor has the capability to reduce raw data from the Sea Lite Beam Director and the Arrow interceptor focal plane array. Other sensor reduction algorithms can be incorporated as desired. We will describe our approach in developing ThOR and Advisor. We also will show ThOR simulations, and test data from SLBD observations of Lance and Storm flights at White Sands Missile Range.

Introduction

The Program Executive Office (PEO) Missile Defense recognized the fact that there was no signature code available that could adequately model the type of stressing conditions that are a part of Theater Missile Defense (TMD). It was decided to develop a modular platform that could use the best of existing codes in an integrated fashion. After review, if it was determined that codes were unavailable or inadequate, codes would then be modified or developed. Since this code would be applied by interceptor systems, it was decided to include the codes that would model effects such as aerothermal heating and seeker effects.

The PEO also recognized another weakness of the available codes, in that there was no sharing of information between an optical and radar (RF) simulations. It is important to share information, particularly shape definition, between the two simulations. PEO wanted a modular platform that could accept RF facet generating models (such as ACAD, BRLCAD, etc.). This process is shown in figure 1. This is important for the PEO since as their systems (THAAD, PAC-3) are designed, developed and tested, everyone should be working with the same specifications.

Theater Optical and Radar (ThOR) Code

ThOR is designed to use a backbone of selected codes, but the option is available to use other codes, as the user sees fit. ThOR is driven through a graphical interface to ease the use of developing a simulation and uses a knowledge based system to check for errors or

omissions before the codes are run. The input was designed also to be specific enough so that the information could be translated for any input deck for any code. The philosophy behind the development of the front-end was to make as easy as possible, while still eliciting all the information needed, allowing the user to spend more time doing analysis instead of developing input decks.

Input Descriptions

The main screen of the input section is shown in Figure 1. Along the top are options of running ThOR or using some of the programs continued in the toolbox. Most of the tools in the toolbox are programs that the user will find useful, such as adding materials to the thermal database, without having to leave the ThOR environment. On the top left portion of the main screen is a status of the input. As the user defines sections of the input, a light is turned on to indicate that that section is completed. The user can also readily tell how far they can run ThOR by what input information is available.

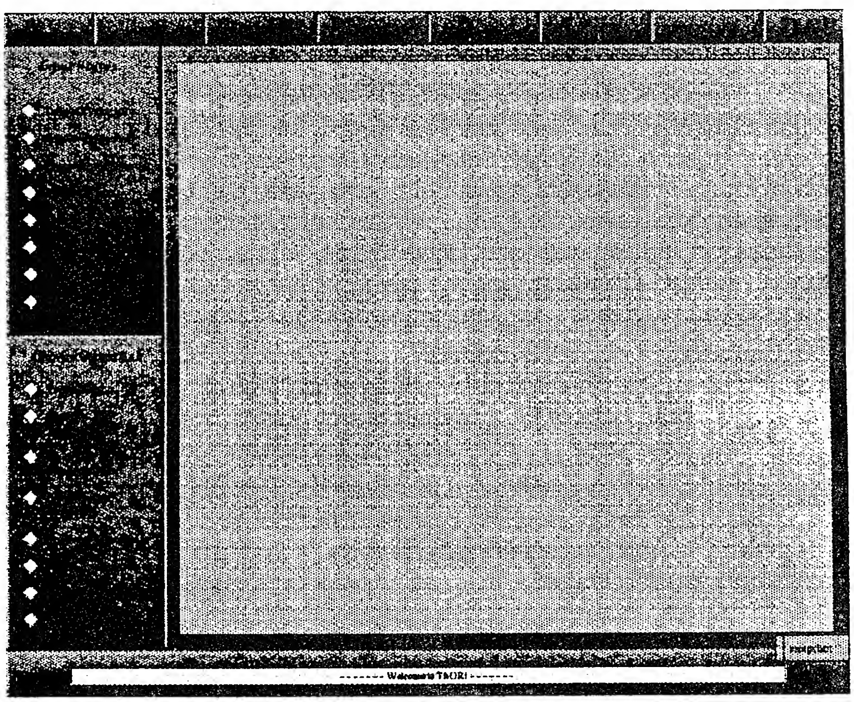


Figure 1. Main screen of input section for ThOR.

In the lower left portion of the main screen is the way that the user chooses output (subsequently, how far ThOR will run). ThOR is a knowledge based system and will inform users that they do not have enough input if they request output that is not covered by their input.

The environment screen is shown in Figure 2. In this screen the user is able to define the type of earth model they desire, the position of the sun, and the type of atmosphere that is needed.

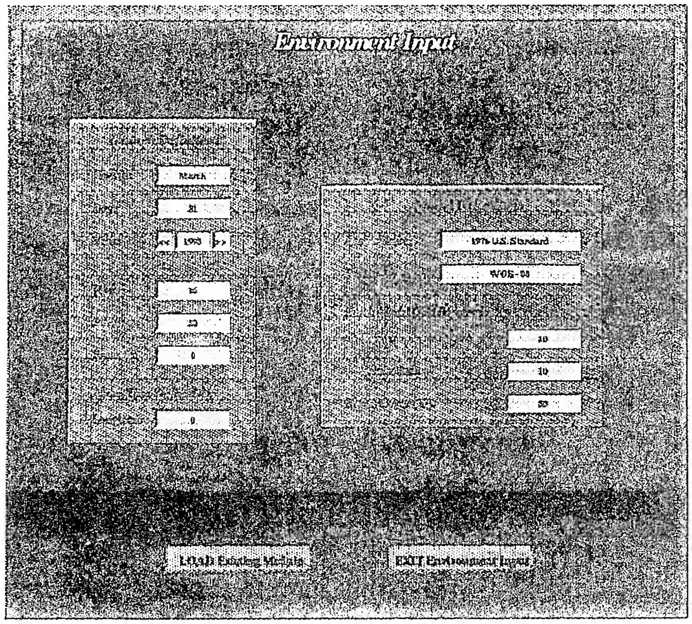


Figure 2. Environment screen to detail required inputs.

The trajectory input is shown in Figure 3. The user has the option of allowing the trajectory code to calculate the ballistic trajectory, or propulsed trajectory. The user can also use data from files that have either been generated by other codes, data from real flights, or from previous ThOR runs. The files can be used to describe a portion of the flight, then the trajectory code can continue the trajectory, if desired.

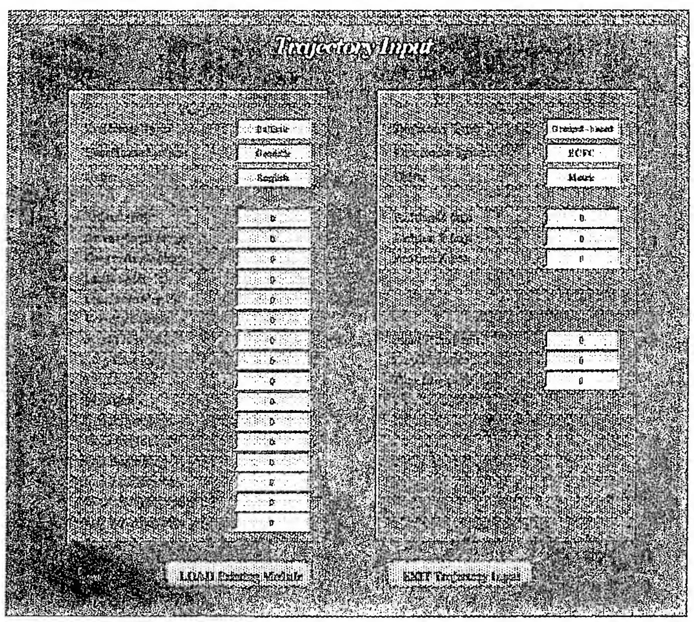


Figure 3. Trajectory input window.

The shape window is shown in Figure 4. The user can develop their target using the resident facet generation code developed by SY Technology, or they can load in a facet model generated by other codes, such as ACAD, BRLCAD, etc. If facet models of RF targets are used, conversions are handled to decrease the number of facets for the optical model. Typically, the optical model does not have to be as detailed as the RF model, but

the choice will be up to the user, depending on the desire for run time or detail. In conjunction with the shape definition screen, is the material identification screen (shown in Figure 5). The user can model various material lay-ups using a material database that is derived from the Optical Signatures Code Thermophysical and Optical Database, and information gathered through the PEO programs.

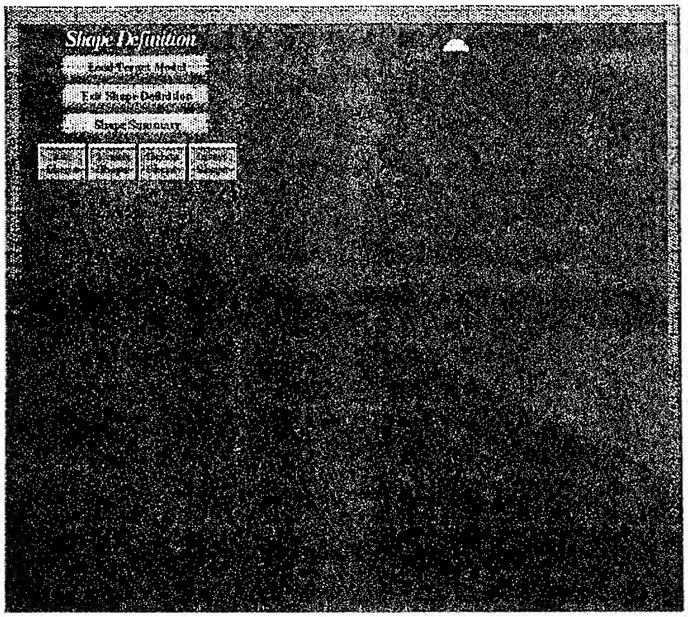


Figure 4. Shape definition window.

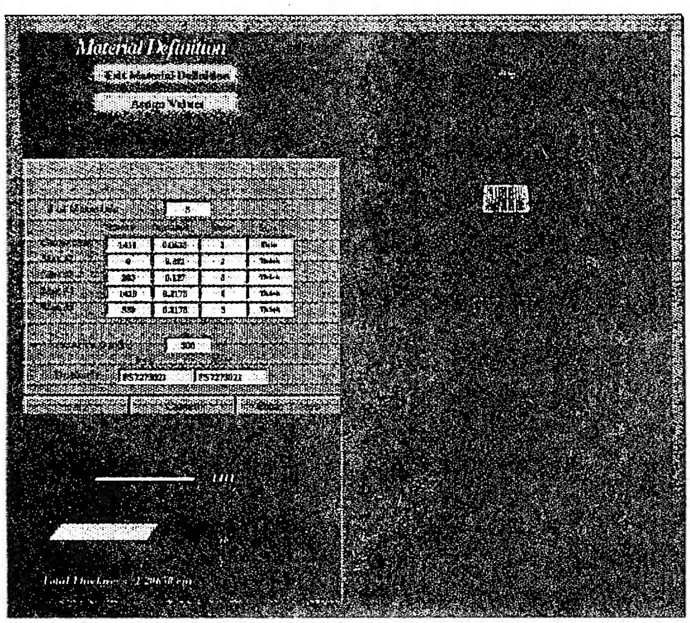


Figure 5. Material definition window.

The thermal window is shown in Figure 6. Most of the information that one would need to run a thermal program is contained in the materials window discussed in the last paragraph. In this window the user can dictate the type of heating that is desired. Currently, the user can chose between running a modified LANMIN exoatmospheric and/or endoatmospheric, or invoke EXOHT3 from the Optical Signatures Code to run in the exoatmosphere only. A

feature that is common among most of the windows, is the capability to read in a file. This is important because it gives the user the flexibility to read in temperatures generated by other programs. It also allows the user to save time by running the simulation once, then simply reading in the temperature file for subsequent simulations.

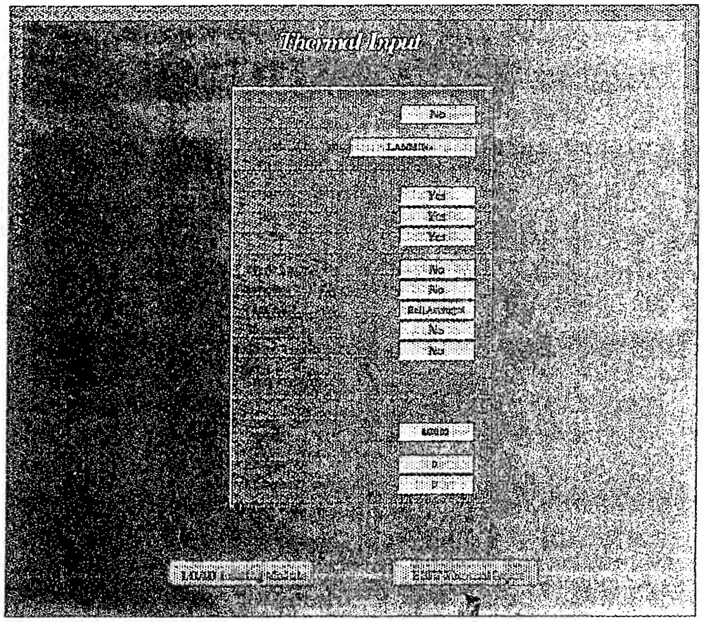


Figure 6. Thermal input window.

The signature definition window is shown in Figure 7. We have designed this window so that the user can have the flexibility to calculate self-emitted and/or reflected components. Being able to calculate only certain components is important for analysis of anomalous phenomenology. This window details the parameters needed for a point-source signature, such as wavebands.

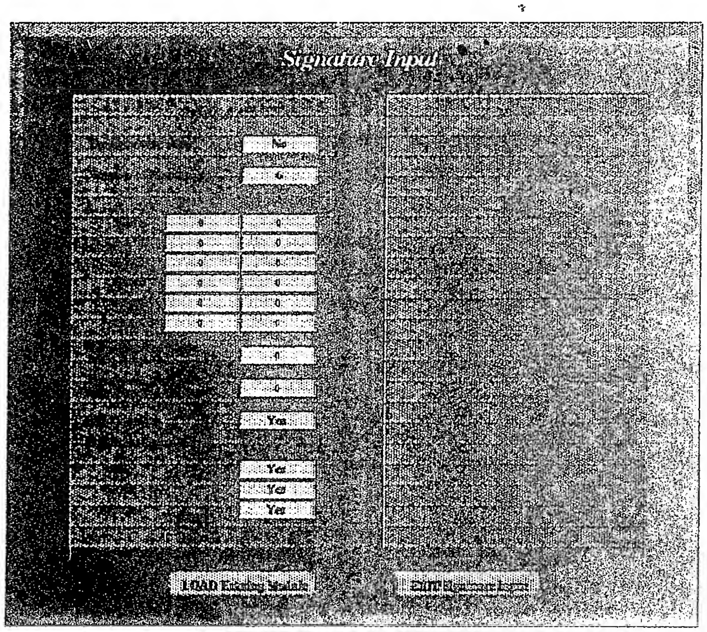


Figure 7. Signature definition window.

The sensor designation window is shown in Figure 8. The user can choose either a resolved or point-source sensor. If modeling an imaging sensor, such as Arrow, THAAD, or SLBD, then the user can detail parameters that are needed for a resolved signature, such as focal plane type (InSb, PtSi, or HgCdTe), focal length, and aperture diameter.

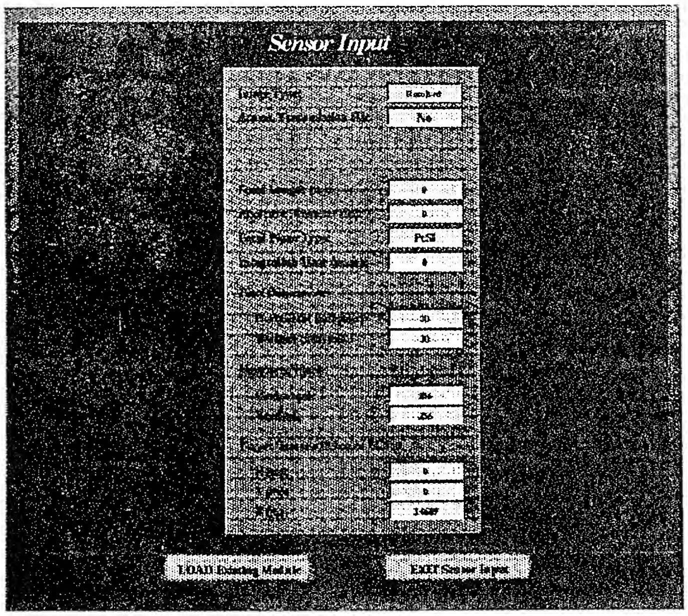


Figure 8. Sensor designation window.

After the user has defined all their inputs, they then choose the type of output that they desire. The output that is chosen dictates the way that ThOR processes to obtain the required data. From the main window, the user then invokes ThOR. All of the inputs are saved to an input file. The flow of this input process is shown in Figure 9.

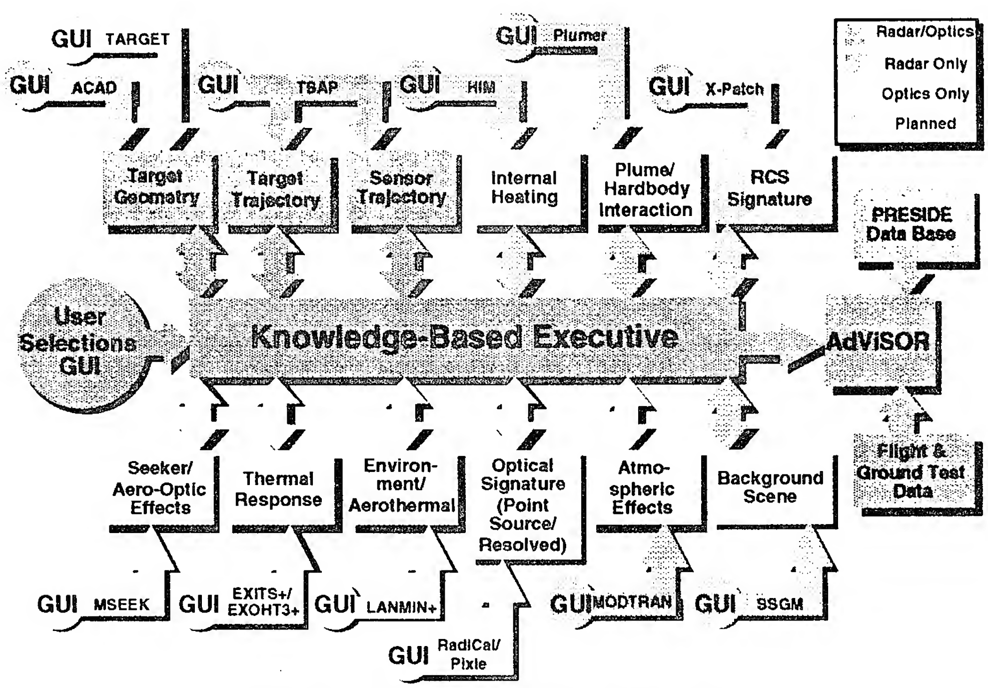


Figure 9. Flow diagram of input process.

ThOR Processing

After the input definition is completed and ThOR is invoked, the input is then read. Depending on what has been written to the input deck, a certain path through ThOR will be followed. A flow diagram of the ThOR processing is shown in Figure 10.

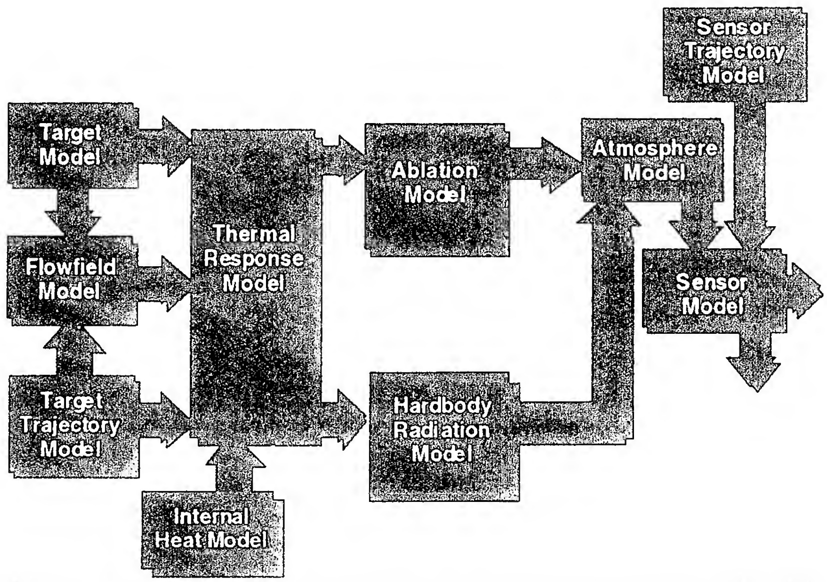


Figure 10. Flow diagram of interaction between codes in ThOR.

Table 1 shows the codes that are available within ThOR. Some of these codes have been modified, such as LANMIN. LANMIN is an aerothermal code that does a particularly predicting the thermal behavior of conical type of bodies. For our applications, we needed LANMIN to be able to handle blunt bodies as well. LANMIN has been modified to correctly predict the thermal behavior along a target body due to blunt nose effects. We also modified LANMIN so that is calculated temperatures by taking into account the effects that the environment would have on a body. A discussion of all the modifications of LANMIN can be found in the ThOR users manual.

Table 1. ThOR modules and status.

Module	Function	Status
TARGET/ ACAD	Shape Input	Developed/ Existing
TSAP	Target, Sensor Trajectory	Existing
X-Patch	Static RCS	Existing
	Dynamic RCS	TBD
	Plume effects	TBD
Lanmin	Aerothermal Model	Modified
EXITS, EXOHT3	Thermal Response	Modified
EXITS	Environment	Modified
IMAGE	Resolved Signature	Developed
MSEEK	Seeker/Window Effects	Modified
AOQ	Aero-Optic Effects	Modified
THORMAT	Thermophysical Data Base	Modified
THORRHOASCII	Optical Properties Data Base	Modified
Advisor	Output Visualization and Data/Simulation Comparison	Developed

We are currently improving and expanding ThOR to include more codes for the user to chose from. We are also continuing to add capability, such as using a static RCS database of a target, and using the same engagement parameters, develop a line of sight RCS signature.

SLBD Data of Storm Flights

We have been using ThOR and ThOR's post processor, AdViSOR (Advanced Visualisation of Simulation or Reality), for the analysis of data. We have good success in verifying the accuracy of ThOR's predictions as compared to data.

One particular data set is from the Storm 8 flight which took place at White Sands Missile Range in New Mexico on 7 February 1995. The RV is a HERA target set on a Storm booster. The target is shown in Figure 11. The trajectory and velocity are shown in Figure 12.

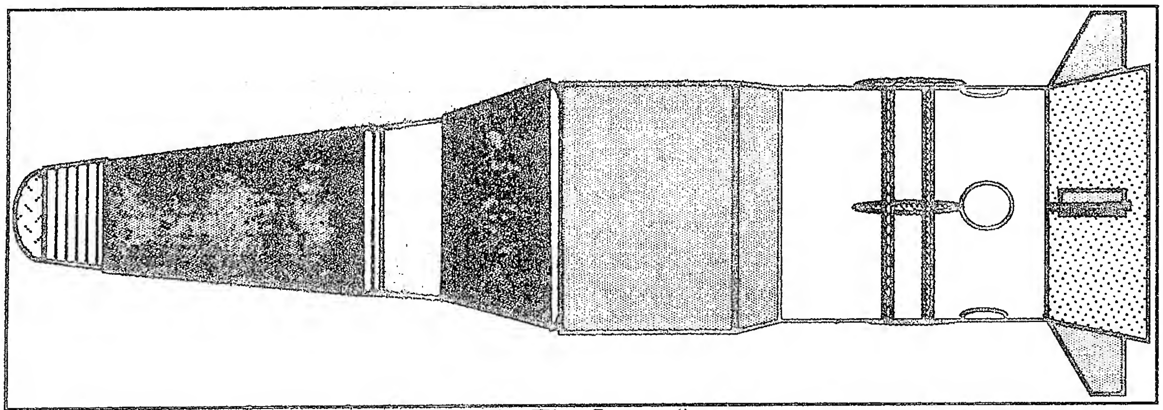


Figure 11. The Storm 8 target.

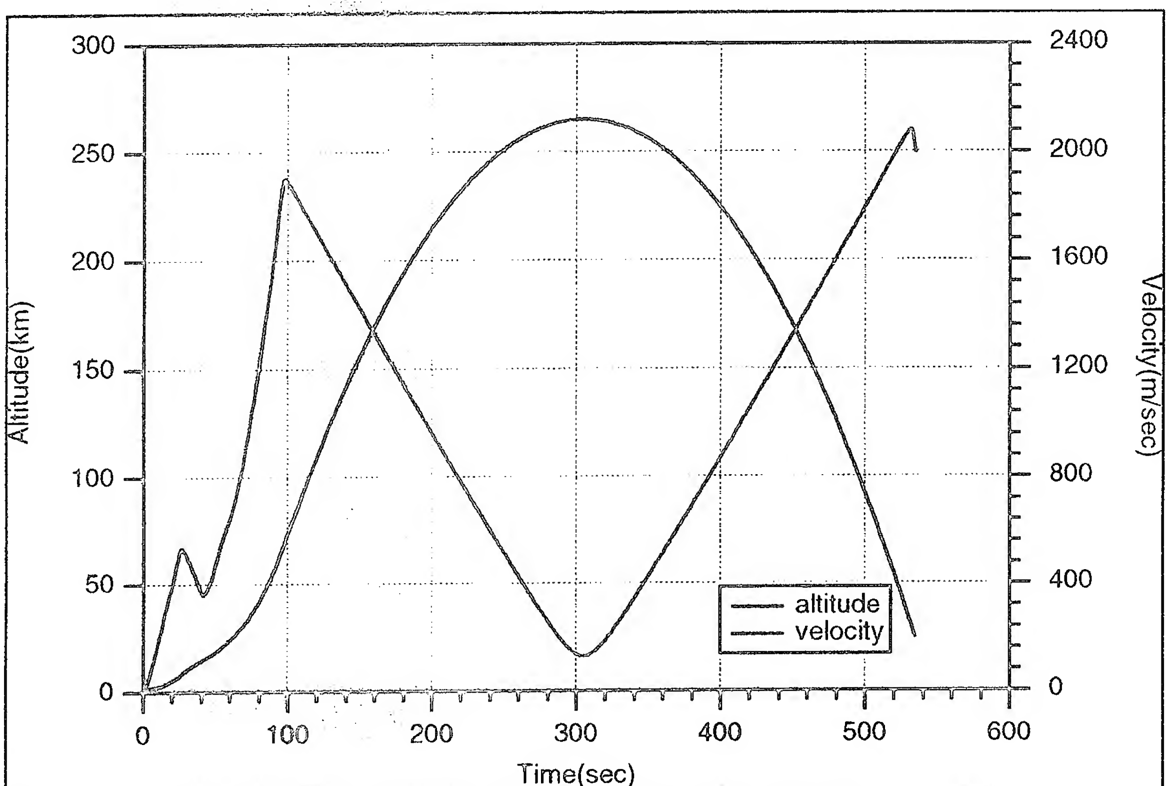


Figure 12. The Storm 8 altitude and velocity.

Several sensors were at this test and acquired data. Included in the sensor suite is SLBD. An image from SLBD of the Storm 8 and an image from ThOR of the SLBD viewing Storm is shown in Figures 13 and 14 respectively.

Figure 13. SLBD image of Storm 8.

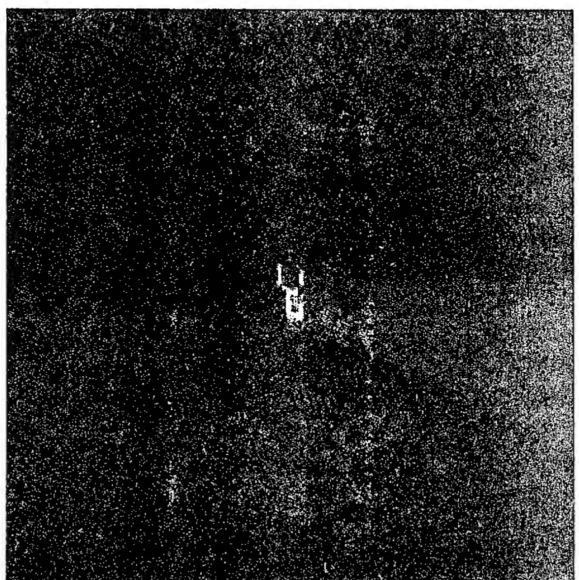


Figure 14. ThOR simulation of SLBD viewing Storm 8.

The target was instrumented with thermocouples, both in-depth and surface. A comparison of simulated temperatures with measured temperatures from various locations on the target is shown in Figure 15.

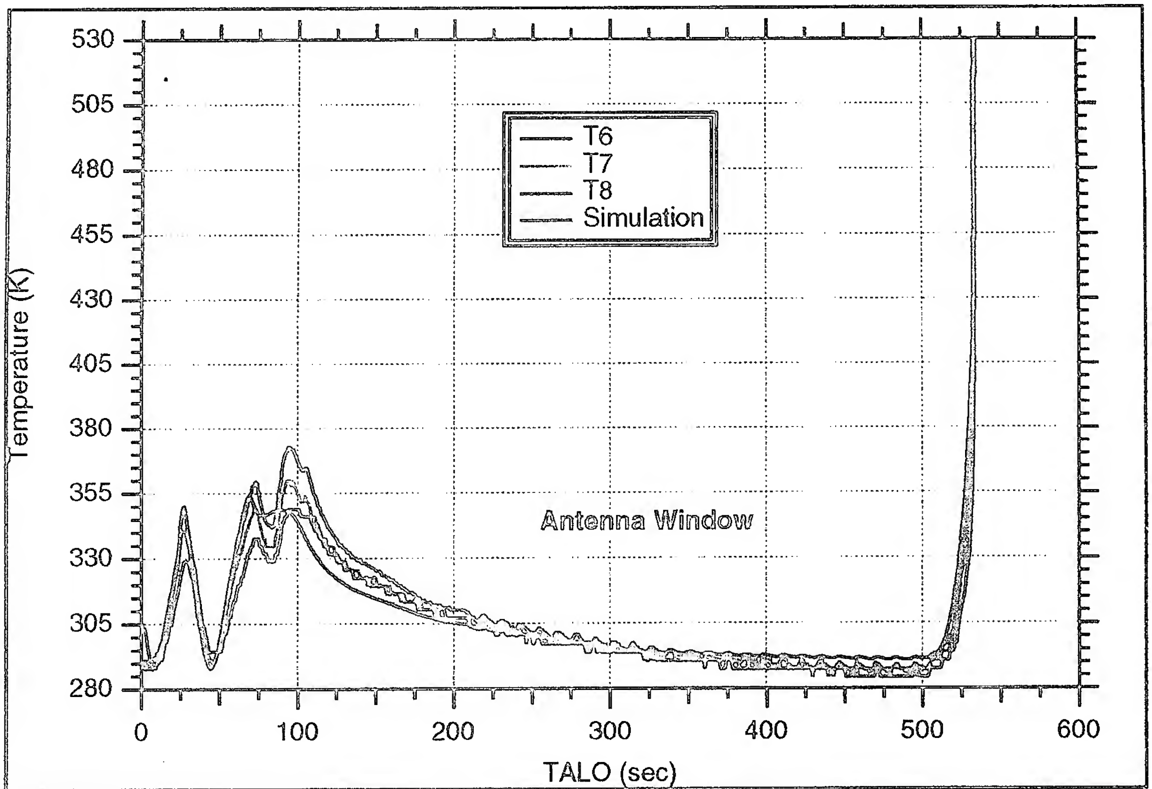


Figure 15.

Further discussions of the validation of ThOR using the Storm data sets can be found in the ThOR manual.

Bibliography

"LANMIN Users Manual", REMTECH, Inc., Huntsville, Alabama, August, 1983.

"Optical Signatures Code, Volume 1 - BASIC Option", Teledyne Brown Engineering, Huntsville, Alabama, March, 1994.

"Optical Signatures Code, Volume 8 - Thermophysical Properties Data Base Handbook", Teledyne Brown Engineering, Huntsville, Alabama, March, 1994.

"Storm 8 Characterization Data Summary Report", SY Technology, Huntsville, Alabama, March, 1995.

"Storm 8 Signature Analysis Quick-Look Report", SY Technology, Huntsville, Alabama, February, 1995.

"ThOR Users Manual", SY Technology, Huntsville, Alabama, April, 1995.

"User's Manual for the Trajectory Simulation and Analysis Program (TSAP)", Sandia National Laboratories, Alburquerque, New Mexico.

AMBIGUITIES IN THE EW SIMULATION ENVIRONMENT

Paul R. Boehm
Advanced Systems Development, Inc.
Manager, Pacific Operations
San Clemente, California 92673

1.0 INTRODUCTION

Ambiguities can be considered one of the most costly problems in the EW community. Ambiguities can add to testing time, increase data reduction time, and mislead engineers in diagnosing problems. Training on a range can be for not if the lab trainers respond differently to the real world. If the simulators and/or trainers do not provide the proper fidelity the Electronic Combat Equipment may not recognize it or it may react differently against a real system and this can lead to the loss of life.

Ambiguities appear when the EC equipment moves between lab, flight-line, test range, and the trainers. Ambiguities fall into many classifications. Some ambiguities are unavoidable, when they arise contingency plans need to be made and all the User's notified of the problems. With this notification several User's will be aware of it and the trainers and other equipment can account for it. Many ambiguities can be avoided from the beginning. If a detailed analysis is made and the potential problem is recognized in the early planning stages of a project, adjustments can be made. Of these potential problems, the following have been identified by various sources as ambiguities that continuously appear.

1) Platform and EMI effects
2) Selected RF source mismatch
3) Pulse Shaping
4) Pulse train modeling
5) Dynamics modeling
6) Multi-player interaction.

1.1 PLATFORM AND EMI AMBIGUITIES

One ambiguity found between laboratory testing and test ranges is due to the platform the Electronic Combat Equipment is mounted on. The most common of these is EMI but, the platforms structure can also come into play. The EMI is on the top of the list, since EMI characteristics can not be easily simulated and are not present in a lab environment. This is because not all of the electronic equipment is present and/or is not in a normal operational mode. For example, transmitters are off or being injected into dummy loads. The simulation of EMI is difficult due to its characteristics in general.

Testing in an anechoic chamber can increase the EMI effects since the actual platform with all it's electronic equipment is present and most of it active. However, there will still be unknowns that are not present since this is a "clean" environment. For example, since engines are not running in the chamber, platform power is supplied from either a support cart or some other external source. It is possible that the prime power is conditioned better than the actual on board generator(s). Another factor left out is the ever present, real world, EMI. By this I am referring to that produced by power lines, radio and television stations, and other such devices. More than

one flight test failure was caused by these and other EMI effects. Undetected EMI effects have been responsible for the loss of aircraft and lives during test flights and operational training flights such as the initial loss of some helicopters from the EMI transmitted from high tension power lines.

Platform structure is another effect that can only be partially checked in a laboratory and in some cases even in an anechoic chamber. The most common of these platform structure problems concerns the placement of antennas. The poor placement of transmit and receive antennas can cause interference between various onboard systems. Another can be a combination of the platform, external stores, pods, and the antenna characteristics. These can cause blind spots that can greatly reduce the capability of the Electronic Combat Equipment. Still another can be caused by the antenna position relative to the platform engine. This is more prominent in prop aircraft such as the E-2C Hawk Eye, the P-3C Orion, or modified C-130 Hercules, than on a jet aircraft.

On a prop aircraft the propellers can interfere with and/or distort the signal. One such case can be described as "self induced multipath". This is caused by a signal reflecting off the propeller blade and into the receiving antenna. Depending on the traveled distance of reflected signal and the carrier frequency, pulse amplitude can be reduced or even nulled.

In phase interferometer receiver based Electronic Combat Equipment, several effects can take place. The first is dependent on the amount of distortion that is produced. Minimal distortion, such as that caused by vibration or the modulation of the wing may only effect the DF accuracy. If an antenna is mounted on the wing tip, the modulation may not only effect DF accuracy but, can also cause the signal to drop in and out of blind spots. This could effect the mean time to intercept but also give the illusion that the emitter is scanning.

If the distortion is immense, such as that caused by a propeller, the receiver will think it is detecting multiple emitters each with its own AOA instead of a one simple emitter coming from a fixed AOA.

An EMI effect that can be present in a lab, since various equipment is coupled together and may not be present in flight testing, is from 400 Hz noise. 400 Hz noise from the Electronic Combat Equipment can get summed into the 50-60 Hz simulator thereby effecting the integrity on the signal. This is more of a problem on the Rack-and-Stack simulators that are not designed properly and/or recognize that the problem could exist. Rack and stack simulators are those made of Commercial-Off-The-Shelf test equipment such as Sine Wave Generators, Arbitrary Waveform Generators and RF Synthesizers that are put together by a lab. When the rack-and-stack system is connected to the EC equipment, the system's ground and EC equipment ground corrupt each other. In many cases Users do not even recognize the presence of the 400 Hz, or do not understand the effect it may have in deteriorating the simulator signal quality. 400 Hz noise is a big problem with many systems outside the lab also in particular on ships and submarines where several different power sources are used from 28 Volt DC to 440 Volt AC at 50/60 Hz and 115 Volt AC at 400 Hz.

1.2 RF SOURCE AMBIGUITIES

An ambiguity that is difficult to diagnose may be due to the differences in the RF characteristics between an actual radar system, a laboratory simulator, and/or a test range simulator. In general a laboratory simulator's RF is comprised of sources, such as VCO's, and Synthesizers; Attenuators; phase shifters; and pin-diode switches. These components are used to simulate the scanning motion of an antenna; free-space attenuation loss; the characteristics of magnetrons, klystrons, and TWT's, such as rise/fall times and unintentional jitter; and the Angle-Of-Arrival (AOA) of these radars relative to the position of the ownship.

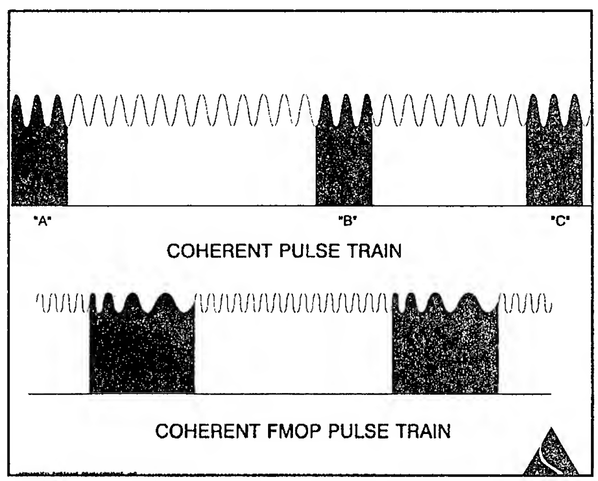
Of all these components, the selection of the proper RF source type can determine the level of fidelity that the simulation will have. Ambiguities due to the wrong RF source are becoming more frequent because of advanced receiver/processors and smart jammers.

If one would look at all of the different radars, one would find that the carriers characteristics can be defined into four classes.

1) Coherent Sources
2) Coherent Sources with modulation
3) Non-Coherent Sources
4) Non-Coherent Sources with modulation.

1.2.1 COHERENT SOURCES

The term Coherent, when referenced to RF Sources is one of the most mis-used and misunderstood terms in EW today, particularly in simulation. By definition, a coherent means "pertaining to waves that maintain a fixed phase relationship". Figure 1.2.1-1 is an example of a coherent pulse train. Normally the term is associated with synthesized sources. Cavity Oscillators, VCO's, and Synthesizers if tuned to a single frequency and never retuned or turned off during the test period will create a Coherent signal. It is not unusual for a specification to use the term coherent in reference to the use of a synthesizer. Using a synthesizer for a source only makes it a stable signal. If a synthesizer is tuned elsewhere during the test period it is no longer coherent.



COHERENT PULSE TRAIN
FIGURE 1.2.1-1

To simulate a stable Coherent signal, a phase locked synthesizer will do the job. If a Coherent interpulse or pulse-to-pulse agility signal is required the use of multiple phased lock synthesizers connected to a common reference signal and a Single-Pole-Multi-Throw switch to select the

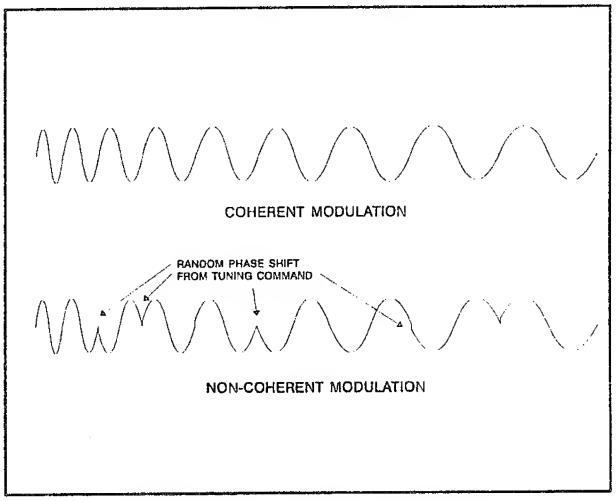
active source is an effective but, expensive approach. The expense is due to the added expense of having to use multiple synthesizers to generate a single emitter. This approach is effective but does have some limitations. However, this method cannot be used to simulate intrapulse modulations due to the inherent design of these units..

1.2.2 COHERENT SOURCES with MODULATION

For a Coherent source that can be modulated to simulate pulse compression, Linear FM, and FMOP, requires a special synthesized source to be designed. This design requirement is one of the biggest challenges in the simulation marketplace. Two important factors must be designed into the synthesizer. The first is the source remains stable and never breaks phase lock. The second is the phase must remain continuous (constant) even during the modulating period and the return to the start frequency.

In Figure 1.2.2-1, one can see the waveform of a coherent FM'd waveform. As frequency changes the phase stays constant. If one were to use a VCO, DTO, phase-locked loop synthesizer, or Yig tuned oscillator, and add a modulation, the waveform would look like a non-coherent source. This is due to the basic design characteristics of these type sources. For instance a VCO or DTO, have incidental FM'ing period that fails the stable requirement but, also when a new tune command is given, the voltage change is a rapid, step change rather than a smooth, controlled, linear change.

For Phased-Locked-Looped synthesizers the loops are unlocked and the synthesizer output looks similar to the VCO. The problem with these phase changes is they may be interpreted as a phase modulation characteristic, such as a type of Barker Code. In other cases this may corrupt a phase modulation code that one is applying to the signal. These are more evident to a piece of Electronic Combat Equipment that has Digital RF Memory (DRFM) technology as part of the system. Another problem is that the incidental FM'ing may confuse the jamming logic and make it difficult to preform a deception technique such as velocity gate pull-off.



COHERENT vs NON-COHERENT
MODULATED WAVEFORM
Figure 1.2.2-1

For a proper modulation to be simulated a Fast, Direct Digital Synthesized (DDS) unit is required. This DDS can be used to replace the phased lock loop circuit of a normal synthesizer. The problem with DDS's are that 8 bit DAC's spurious are to high and corrupt the primary signal. Twelve bit DAC's that can operate at 500 MHz or higher clock speed are required to meet the spurious and bandwidth requirements of the modern EW environment. These characteristics are the goals of several synthesizer houses today and are in the works, with progress being made everyday.

1.2.3 NON-COHERENT SOURCES

Non-coherent sources have no special requirement except for stability and shadow time. The use of non-coherent sources is very popular in multiplexed simulators. Because of this several options are a available to the User. The most common in lab simulators is VCO's or DTO's. These sources are the least expensive and in may ways provide a better capability for less sophisticated equipment. Fast tuning synthesizers such as Comstron or the equivalent may be used instead of VCO's if stability is a requirement. The problem is that fast tuning synthesizers are expensive.

1.2.4 NON-COHERENT SOURCES with MODULATION

Non-coherent sources that can modulate are common. Fast tuning VCO and fast tuning synthesizers are the most popular type of modulating source. These sources are capable of multiplexing emitters, providing broad band interpulse agility capability, simulating high incidental FM'ing Characteristics and intrapulse characteristics such as Chirps. The faster the settling time and the higher the resolution the more linear these signals will be. The problem that can occur with this type source is, they have a tendency to overshoot, prior to settling at the proper frequency. If this happens during intrapulse modulation the signal will appear noisy and non-linear. The sources must have a damping system that allow maximum speed with minimal overshoot. The capability to correctly handle overshoot comes from years of experience and is not normally found in off-the-shelf-sources.

1.2.5 SELECTION OF THE PROPER SOURCE

The selection of the proper mixture of sources is based on several issues. The first of these issues is to understand what type of EC equipment is being tested. If the EC equipment has limited receiver capabilities such as crystal video systems, it is more cost effective to use VCO based sources.

The second is to understand the characteristic of the emitters that are going to be simulated. A problem that has been noted in flight testing against actual systems and laboratory simulators is the differences of the carrier itself due to the type of RF source selected.

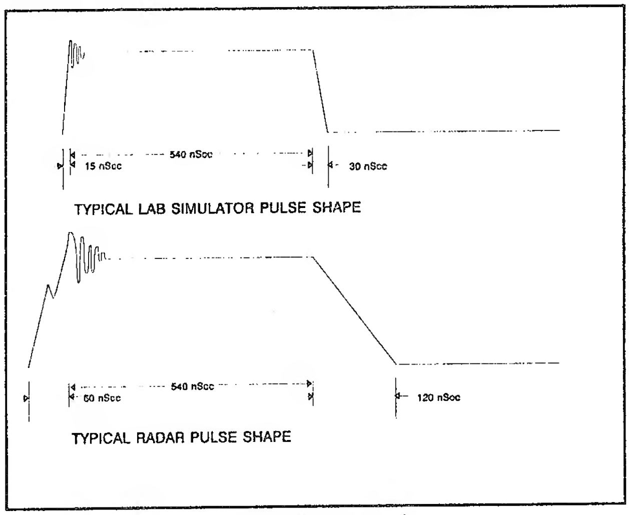
Many labs, in an effort to save money, will buy off the shelve synthesizers and arbitrary waveform generators to make a "home brewed" simulator. In many cases this will be sufficient to do basic testing against Electronic Combat Equipment but, it costs more in flight test time trying to resolve the lab test versus flight test results.

One such problem related to this was where an EC receiver would work in the lab against a "home brewed" simulator and fail flight test. After several failed tests, the User rented time and brought the EC receiver to a facility that had a full up simulator. The problem was found in less then 2 hours. The "home brewed" was synthesizer based, the radars that it failed against where older magnetron type radars that had a large incidental FM characteristic. The simulator emulated these characteristics and the problem was seen for the first time in the lab. It ended

up that the synthesizers were too stable for the simulation. When the system went up against an unstable source it did not know how to track it.

1.3 PULSE SHAPING

Pulse shaping capability is becoming more important as more channelized receivers are being installed in all types of platforms. A characteristic that is becoming more common in ELINT receivers is fingerprinting the pulse for future reference. If a pulse is fingerprinted and associated with the exact platform that generated it, one can then use this data to not only identify the radar type but also the serial number. This can be valuable for multiple reasons including Battle Damage Assessment. Without this capability, if a strike is made against a radar on one day and the same type is active the next day in the same area, one would assume that the strike was unsuccessful. If the radar was fingerprinted before the strike and then checked the day after the strike, one can assess if it is the same radar or a new one.

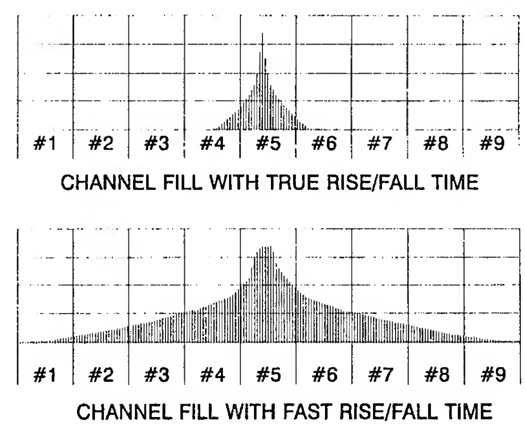


PULSE SHAPING
FIGURE 1.3-1

In dealing with rise/fall time ambiguities these are primarily related to channelized systems. Figure 1.3-1, shows the differences between a pulse generated by a simulator and one generated by a typical radar. As on can see the lab simulators pulse has fast rise and fall times, typically in the 10-20 nSec range, with very little fingerprint characteristics which is the characteristic of a pin diode switch. Because pin diode switches are known to be used, many times Users will specify a fast rise/fall time, thinking of how the receiver would detect it easier without a full understanding of the consequences. In the past some User's have recognized the problem but back then the technology was such as controllable rise/fall time switches where not available. Today several methods exist or are near term to handle this problem.

If one did an analysis of all the emitters in one's scenario, one may find that rise and fall times are actually in the 30-1000 nSec range and have very distinguishing characteristics because of the technology used to create them. Most of the characteristics is do to the thermeonic device that is used. This difference in rise/fall simulations can greatly effect how well an Electronic Combat Equipment reacts to the environment.

If one would compare the spectrum bandwidth of a slow rise/fall time with a fast rise/fall time, Figure 1.3-2, one would observe that the fast rise/fall time emitter would have a spectrum spill over into more channels then normal, thereby reducing the probability of intercept of other emitters that may fall into these side channels. This difference can cause orders of magnitudes in sensitivity loss thereby decreasing the actual pulse density of the lab simulation versus the test range scenario. This differences can be the delta between a piece of Electronic Combat Equipment handling the environment or going into saturation due to the high pulse density and failing.



CHANNEL SPILL OVER EXAMPLE
FIGURE 1.3-2

Another problem that fast rise/fall times can cause is the creation of addition signals by the receiver. If one were to amplitude modulate the spectrums in Figure 1.3-2 relative to the scanning motion of the emitter, one may find that the signal in channel #2 is only present every 5-8 seconds for 6 mSec. To a receiver this could indicate a signal 30 MHz or more away and try to process it. This is especially true for an emitter that can have multiple simultaneous RF beams.

1.4 PULSE TRAIN MODELING

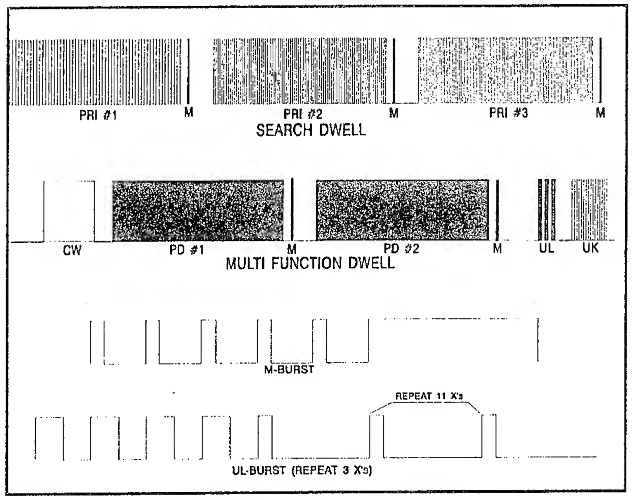
The modeling of pulse trains is an area where most Users make many assumptions that end up playing total havoc with a lab scenario or a flight test. This is also a prominent area where commonality is vital but is the most often overlooked. Some of the blame for this can be put on the End User. This is because this User may incorporate the receiver limits as the specification for the simulator. In doing this the User has automatically asked for ambiguities to be part of the testing cycle. One of the most common specifications that is ignored is with clock based radars. A User may specify that the PRI resolution shall be 100 nSec or 50 nSec which equates to a 20 or 40 MHz clock. These limits once again are typically based on the capability of the receiver or conventional technology.

If one would analyze the PRI values of the radars, one would see that many are based on fixed frequency clocks. The most typical values of older systems are in the 150 to 600 kHz range. This in turn means that the PRI can only be a multiple of that clock value. As an example, if a radar had a 150 kHz clock with a PRI countdown of 53, the PRI would be 353.33333 uSec. With only a 100 nSec resolution an error factor has already been added to the lab simulation versus the test range. The same can be said for new systems, again they are clocked based but, with faster clocks making the error factor even larger.

In cases where high powered transmitters are used on the range versus a replicator or real system this error may be a factor that has been added to the testing environment but, not the real environment. This means that the first time this ambiguity may appear, is in a critical situation.

Another problem that is waiting to arise is when a User, after noticing that the EC equipment reacts the same against a radar in a one-on-one test mode with or without a pulse train feature present, decides to simplify the test to save resources. In these cases, the User may not even bother to add the characteristics on to the simulated emitter. In most cases this will come back to haunt them at the most inopportune time. What is not asked by the User is, " What happens when multiples of this or another emitter appears with this emitter?". A User must always remember that a pulse, even though it is not detected by the receiver, is present in the receiver and if a pulse is present in the receiver, the time that it is present may effect another pulse that can be detected.

For an example of this look at figure 1.4-1. These pulse trains are appearing at various times in association with an electronic scan dwell. A receiver may not react any differently with "M" Burst programmed into a emitter or not. This lack of an "M" burst may be due to the limitation of one's simulator such as the number of complex signals that can be done or for some other reason. The problem is that the pulses of "M" burst will occupy a period of time in the receiver of 25 uSec. In a multiple emitter environment that occupation may effect the detection of another signal. This is especially true when more than one of a radar type is present. The amplitude of this burst may be higher than that of a detectable pulse, thereby creating a pulse-on-pulse situation that the receiver may not handle in a multi-emitter environment.



COMPLEX PULSE TRAINS
FIGURE 1.4-1

1.5 DYNAMICS MODELING

The addition of dynamics to any test can complicate matters and add ambiguities that were never considered. These ambiguities can be the result of power changes due to free space attenuation, mode transitions, multipathing (particularly in over the water test), atmospheric conditions, and receiver antenna pointing position relative to the platform.

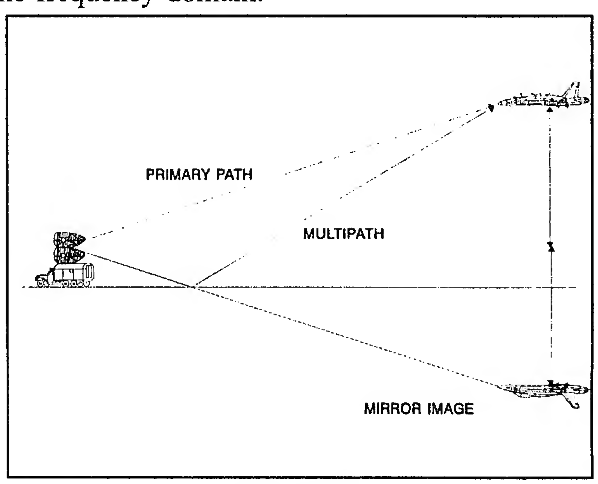
Free space attenuation effects are probably the most common of these type problems. Most simulators create signals based on a two dimensional plane instead of three dimensional. To add to this problem the target position in the scan is fixed. What this means is that the main beam of the threat is always on the ownship platform. In reality, it is possible for the platform to fall into sidelobes and nulls particularly in the elevation plane of a scan. This can effect the detection range of the receiver or depending on the flight profile cause the signal to drop in and out.

Another ambiguity that is related to free space attenuation can effect smart/power managed receivers. These systems track the amplitude of every pulse from an emitter to determine the jamming strength and extract parametric information. With this information the system can determine the scan period and even the multiple periods of some scan types such as Bidirectional sector and Raster scans. To perform this function the EC equipment must have a defined logic as to determine from sweep to sweep what is the most powerful pulse.
In a non-dynamic scenario environment this is not a difficult problem since many variables stay constant. In a true dynamic scenario environment nothing is constant nor can be 100% predicted. If one were to do a quick analysis of all the variables and conditions that determine if an emitter can be intercepted and processed, one could see that a poor dynamic simulation can cause a magnitude of ambiguities. Consider the following simple scenario, a circular scanning radar whose antenna is 15 feet Above Ground Level (AGL). The ownship is an aircraft traveling at 450 knots at 250 feet AGL in a terrain following mode with an amplitude comparison receiver.

If one where to look at this in the simplest sense, the obvious variables that would change would be the received amplitude of the pulse due to the scanning motion and to free-space attenuation as the aircraft to emitter range changed. Another factor is the initial detection of the emitter due to the horizon created by the curvature of the earth and terrain characteristics sure as mountains and hills. These are basics that almost every trainer, lab simulator, and mission planner look at.

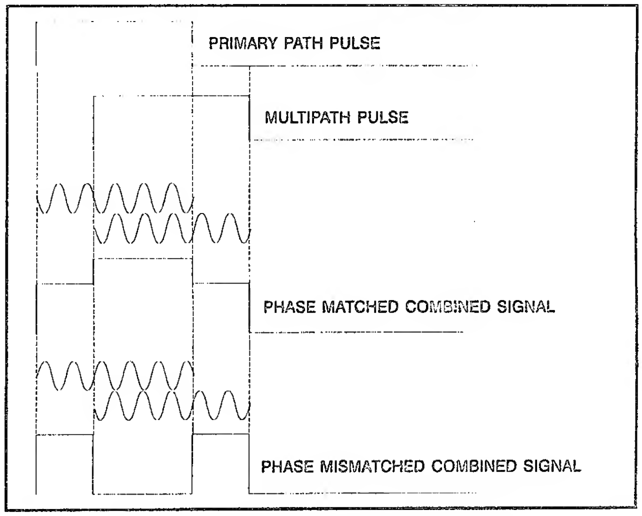
Unfortunately, for the EW community as a whole, life does not consist of simplistic solutions, especially in the simulation world. To even come close to a test flight replication numerous other factors come into play. By applying six degree freedom of motion, the receiving antenna characteristics becomes a major factor. The banking motion of an aircraft can blind the receiver by pointing the antennas away from the transmitter. Because of this problem, the receiving antenna must be simulated in three dimensions. This simulation must not only be angular sensitive in azimuth and elevation but also in the frequency domain.

Multipath can also effect a receiver in many ways. Multipath can increase the strength of the signal the receiver detects or reduce it to practically nothing. If one broke down the major components of a multipath signal, one would find that it contains its own elevation Angle-Of-Arrival and maybe an azimuth AOA also. It has it's own Time-Of-Arrival and its own amplitude relative to the reflective quality of the reflecting surface plus, free space path loss. The combination of these variables mixed with the true path signal can create unusual signal characteristics as shown in Figure 1.5-2. To simulate these effects correctly an extensive amount of additional computing hardware and RF hardware is



MULTIPATH
FIGURE 1.5-2

required. The normal signal path needs to be split off into two separate paths. The one will be the normally modulated signal. The second which is split after the modulation has been added is then time delayed, attenuated, and phase shifted to reflect the multipath signal characteristics. This signal path is then summed into the normal path signal, thereby creating a true signal, Figure 1.5-3. This simulation is easier over water than over land but still requires a large amount of processing power to compute these characteristics in real-time. With an over the land scenario the reflectivity constantly changes due to the composition of the terrain. Repeating a test with the same results on a test range can be difficult due to the use of expendables. Each time an aircraft approaches an emitter and pops chaff the multipath factor and test results change. In the late afternoon when the wind increases the chaff is scattered and a new multipath factor is in effect. Other effects such as wind, rain and clouds can also effect a flight test.

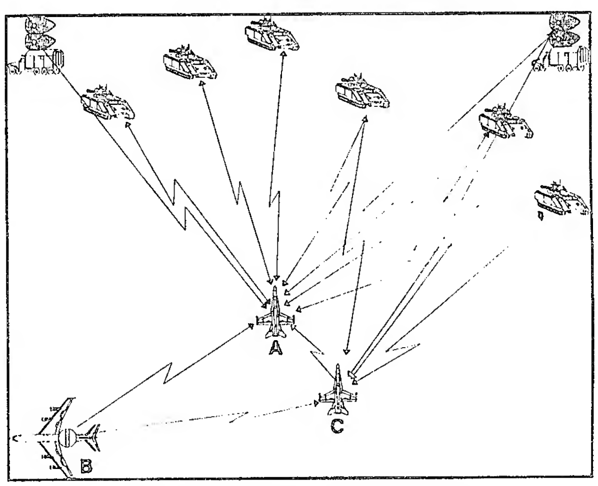


MULTIPATH EFFECTS ON A PULSE
FIGURE 1.5-3

The most critical factor that effects every aspect of the test is the human factor. With the number of personnel required to conduct a flight test this becomes even a bigger factor. Lab simulators require the capability to add these random factors into a test to get good results. One method is by adding multi-player, man-in-the-loop, consoles to a simulator to allow outside intervention of the environment.

1.6 MULTI-PLAYER INTERACTION

Multi-player interaction is one of the most forgotten effects in all test results, training missions, and real world situations. Most lab systems, trainers, and even flight tests only concern themselves with a one player environment. The problem is that unless that is the normal mode of operation one will not truly understand the capabilities of ones Electronic Combat Equipment.



MULTI-PLAYER INTERACTION
FIGURE 1.6-1

Self protect jammers and a flight's own radars can also create their own problems. If one looks at the determining factors for an RWR or jammer to process an emitter it looks at PRI, Frequency, and AOA as a minimum to categorize a signal for tracking. In an EW

scenario where jammers use broad beamwidth and broadband jamming antennas jamming signals are transmitted all over the environment, multi-player interactions take place. If one where to look at the placement of emitters in Figure 1.6-1, one can see that one wingman's jammer could be transmitting into another wingman's systems and using resources. To a jammer and or receiver, the jamming signal from aircraft 'C' can be triggering aircraft "A". To make matters worse is that aircraft "A's" jammer would be using valuable assets to jam what appears as a lethal threat but in reality is not lethal to 'A'. In return, 'A' signal could be effecting "C's" signal. Another is when a flight group begin activating all of their radars when weapons delivery is commencing. The worst case of this type of interplay that has been documented is, that some countries have everyone in a flight group operating at all times. The reason for this has wide speculations but, the fact is that it wastes each others jamming and receiving resources that may not be present during normal testing and training. In meetings and discussions with test range and analysis personnel, they have noted that more flight test failures have happened when multiple players have been introduced.

A multi-player characteristic that is becoming more prominent concerns multi-tracker, multi-function radars that adjust to the environment, such as the Former Soviet double digit systems, the Patriot, and SPY-1. With these type of radars, the number of players effects the results that you see. These radars, not only effect lab simulation, trainer, and flight line testers, but, they effect the test range simulators.

To accomplish this capability, these radars use such technology as slotted array or space feed array antennas. These systems use phase controllable elements to electronically point the beam instead of physically moving the antenna. These system's can reposition their mainbeams on any target within microseconds. This repositioning can decrease or increase the visit time that the Electronic Combat Equipment sees the signal and also increases the chances to multiply jammer interaction, especially among wingmen.

Present range simulators such as the MUTES, Mini-MUTES, and AN/FSQ-XX with their conventional thermeonics devices, electro-mechanical positioners and antennas, cannot reposition fast enough to simulate these radars. In many cases these systems are unable to keep up with tracking a single high performance target. To assist in this problem these systems use broad beamwidth antennas. The problem with this is that several aircraft can be stimulated at the same time causing false test results. To simulate a multi-tracking radar system one really only has three choices;

1) Go with the present convention and accept the results will be flawed.
2) Use a system that has multiple antennas that can be pointed independently.
3) Use an electronically steerable antenna.

Obviously, the second and the third solution are a more expensive method but, will provide the most accurate environment, depending on the technology used, and can have a long term cost savings. The decision on what method is to be used must be weighed based on the End User requirements and on what the final goal to be accomplished is.

The state-of-the-art in electronically steerable antennas is solid-state, phased array antennas based on Microwave Monolithic Integrated Circuits (MMIC) technology. This technology is being employed in radar such as the USAF F-22 Advanced Tactical Fighter and expendable decoys such as Texas Instruments Gen-X units. MMIC technology in radars and for test range emitters provide a capability never seen before.

As previously stated, this problem is not limited to the test range system, it is also a problem in lab trainers. The difference is that test range systems have physical limitations such as transportability, the amount of air space and land available, the cost versus the number of emitters that can be generated. Labs simulators have other limitations that the range do not have such as EMI effects.

It is widely believed that the most economical way to build a trainer and provide a solid simulation is with a software intensive system. A large percentage of all trainers are software based. The complete receiver and threat models are emulated in software. The problem is that one has a software model that interprets what the actual Electronic Combat Equipment software is telling the hardware to do. This in itself means that you will have a faulty model. Software based systems have several inherent problems. Some of these are:

1) Most software solutions are designed to model the characteristic of the displays. In may cases the emitter library consists of only the symbols that are to be displayed for each mode of operation. This means that the simulation will not be accurate.

2) If the displays are the primary concern then the models will not be detailed enough to account for pulse-on-pulse characteristics, identification ambiguities, or the saturation of the receiver due to excessive pulses.

3) Problems arise from attempting to completely model an environment in real-time. As previously discussed, the USAF is working towards this with their J-MASS program and at the present time the accurate models can not run in real-time even in a one-on-one scenario.

With hardware based trainers many of these problems do not exist. This is primarily because these trainers use a complimentary blend of actual ECE hardware, simulation hardware with receiver emulations boards, and emulation software.

2.0 CONCLUSION

Ambiguities will always exist in every phase of EW testing. If one is aware of the ambiguities ahead of time and plans for them time and money can be saved.

OPTIMIZED COORDINATION OF ON-BOARD MANEUVER AND COUNTERMEASURE ASSETS FOR OWNSHIP SELF-PROTECTION

George Chapman

Mission Research Corporation
735 State Street, P.O. Drawer 719
Santa Barbara, CA 93102-0719

ABSTRACT

Mission Research Corporation has developed a system, known as the Automated Process of Countermeasure Association, commonly referred to as CMAT (sponsored by the Air Force Pilot's Associate Office and NAVAIR). CMAT is the designation of our approach to threat system situation awareness and reaction strategy generation. CMAT is a complex system that contains an extensive set of integrated algorithms and data representation schemes. The purpose of the CMAT system is to autonomously recommend an optimal countermeasure and/or coordinated maneuver response to the pilot in order to assist him defeat an attacking threat system. The inputs to the CMAT system are the measurements made by the ownship's various onboard sensors which measure the radar, infrared, and/or kinematic profiles of airborne or ground-based threats (offboard sensor data can also be utilized as available). The output from CMAT is the optimal threat response action which is then recommended to the pilot. CMAT uses a variety of artificial intelligence techniques, including fuzzy uncertainty management and mimic net automated learning. These techniques were specifically designed to handle the unique and challenging problem associated with threat response in environments of high data uncertainty.

An important feature of the CMAT algorithm is that it actually calculates the effectiveness of every response option. We purposely avoided a rule-based system in order to maximize CMAT's ability to automatically adapt to new intelligence information concerning the description of known enemy threat systems, countermeasure effectiveness profiles and sensor information. The calculation procedure takes into careful consideration the threat-to-ownship engagement geometry (azimuth, elevation, time-to-go, and launch range), sensor measurement uncertainty, fused sensor data, intelligence uncertainty concerning the operating characteristics of known threat systems, multi-threat scenario effects, and a premission threat danger map which describes spatial likelihoods of known threat systems.

By monitoring the processed intercepts and detections from the avionics, countermeasure reactions are selected and scheduled. By considering all potential threat systems represented by signal intercepts, and determining whether an antiaircraft missile is on the way, the CMAT procedure selects a response to maximize survival prospects considering this and possible future threats to ownship, mission constraints, and aircraft status. Countermeasure selection maximizes the estimated frequency of survival without requiring a threat identification. The entire database of likely threat systems is considered in the reaction selection. The CMAT approach is unique in that it uses a data driven algorithm to accommodate changing information without requiring reprogramming of the system software. Information concerning threat capabilities, intelligence estimates, tactical innovations, signature changes, and avionics sensor enhancements are placed in editable databases which are then used by the CMAT program.

Introduction

Over the past several years MRC has been developing an automated threat response and threat prioritization program for several customers, including the Air Force's pilot's associates office and for NAVAIR. This system, which is called CMAT (Countermeasure Association Technique), is a complex system which contains an extensive set of integrated algorithms and data representation schemes. The purpose of the original CMAT system was to autonomously recommend an optimal countermeasure and/or coordinated maneuver response to the pilot in order to assist him defeat an attacking threat system. The inputs to the CMAT system are the measurements made by the ownship's various onboard sensors which measure the radar, infrared, and/or kinematic profiles of airborne or ground-based threats (offboard sensor data can also be utilized as available). The output from CMAT is the optimal threat response action which is then recommended to the pilot. The system currently consists of over 25K lines of validated Ada code. It uses a variety of artificial intelligence techniques, including fuzzy uncertainty management and mimic net automated learning. These techniques were specifically designed to handle the unique and challenging problem associated with threat response in environments of high data uncertainty.

The objective of the CMAT work has been to design and demonstrate an artificial intelligence system that reduces pilot workload in the areas of threat classification, prioritization, and response recommendation. In our approach we have focused on the following features: system adaptability, system trainability, efficient data access, accommodating system growth, efficient data compression, and uncertainty management.

By monitoring the processed intercepts and detections from the avionics, countermeasure reactions are selected and scheduled. By considering all potential threat systems represented by signal intercepts, and determining whether an antiaircraft missile is on the way, the CMAT procedure selects a response to maximize survival prospects considering this and possible future threats to ownship, mission constraints, and aircraft status. Countermeasure selection maximizes the estimated frequency of survival without requiring a threat identification. The entire database of likely threat systems is considered in the reaction selection.

The CMAT system concentrates on achieving three functions:

- quality assessment, fusion, and processing of the sensor data available on tactical aircraft
- recognition and priority assignment to threat situations requiring response
- recommendation of viable defensive reaction given the threat situation

The CMAT approach is unique in that it uses a data driven algorithm to accommodate changing information without requiring reprogramming of the system software. Information concerning threat capabilities, intelligence estimates, tactical innovations, signature changes, and avionics sensor enhancements are placed in editable databases which are then used by the CMAT program.

Natural language and graphical database interfaces have developed that allow the extensive database information to be displayed and edited with great efficiency. These interface tools enable the CMAT system to be readily tailored for aircraft, tactics, threats, and theater as required by the Army, Air Force, or Navy. This adaptive, data driven approach also has the benefit of removing the CMAT system from the debate over what reactions and tactics are the proper ones. The proper responses and the corresponding effectiveness estimates are added to CMAT by the ultimate users using the database interfaces.

Another unique feature of the CMAT development is the use of efficient data compression and fusion techniques to manage the large amount of expected measurement data. The CMAT system uses quantitative discrimination to associate measured data with stored data, but represents this data in terms of fuzzy sets to manage data uncertainty. Finally CMAT uses a unique MRC-developed tool, called the *mimic net*, to provide threat prioritization and recommendation of optimal threat response. The mimic net is adaptable and trainable like a neural network. It is able to quantitatively identify through off-line training, the various costs and benefits associated with candidate threat response options.

The products delivered to the Air Force at the end of the Small Business Research Program (SBIR) Phase II program included a final report as well as the proprietary CMAT algorithms, which were implemented in over 25000 lines of ADA code. These algorithms include the interfaces to the natural language database, the countermeasure/maneuver effectiveness database, and the missile launch envelope database. This work has been extended by NAVAIR under a new Phase II SBIR effort. Under this new program, MRC is incorporating the CMAT algorithms into the Navy's SH60B rotorcraft platform. We are also identifying the requirements to retrofit CMAT to additional Navy Platforms, including the V-22, H-53 and F-18.

The major functional components of the CMAT procedure are:

1) the **sensor post processor** (performs multi-sensor data fusion, trackfile sort, search and binning),
2) the **chalkboard memory manager** (manages missile kinematic and radar signature trackfiles, performs trackfile fusion, temporal trackfile management),
3) the **survivability estimation module** (calculates countermeasure/ maneuver survivability estimates, considers response effectiveness against multiple threats), and
4) the **resource optimization module** (uses mimic net training procedure, down selects optimal response option considering mission objective, mission phase, costs of available resources).

These four basic CMAT functional blocks are shown in Figure 1 and explained in the subsections below. The key CMAT system features are summarized in Table 1.

CMAT Component Descriptions

Sensor Post Processor. The purpose of the sensor post processor is to store, fuse, and sort measured datasets (trackfiles and/or contact reports) collected by onboard sensor systems. We group these datasets into two types: radar signature data or missile IR/kinematic data. Each dataset consists of measured threat attribute values and their associated measurement variances.

Typically, there will be few, if any missile trackfiles, corresponding to few missiles in flight in the battle area. In contrast, there could be thousands of radar signature datasets, corresponding to all active radars as well as spoof emitters, acquisition and tracking radars, and search radars. The signature datasets enable threat identification and permit appropriate threat response strategy. However, the radar signature datasets must also be associated with the missile datasets. This is especially difficult when angular resolution is low and there are great numbers of distinguishable radar datasets. Within CMAT, two data buffers are maintained, one for each dataset type. Specific algorithms (threat prioritization, or threat response recommendation) join the two datasets according to their specific needs. This architecture minimizes errors that would result if radar datasets were mistakenly associated with missile datasets.

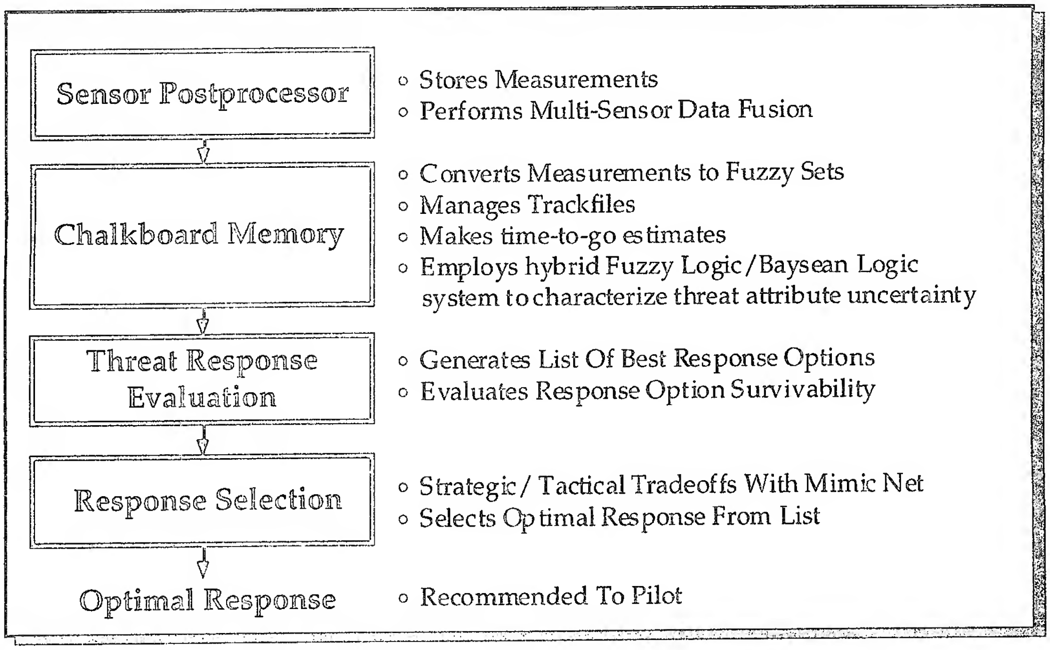


Figure 1. Primary CMAT Functional Blocks

Table 1. Key CMAT System Features

-Data driven	Does not require reprogramming for new threat environments.
-Mimic Net	Exactly mimics expert's selection criteria for threat prioritization and threat response recommendation.
-Fast, Efficient	Uses data compression, graphical database access, analytic computations, parallel architecture, neural processors.
-Adaptive	Readily retrains to changing data. Adapts to expected threat densities, mission objectives, and pilot preferences.
-Manages Uncertainty	Accommodates WARM threats, uncertain intelligence, sensor data corruption.
-Multiple Threats	Recommends effective response against multiple missile launches
-Joint CM / Maneuver response	Recommends a combined countermeasure / maneuver response which maximizes joint effectiveness while minimizing resource cost considering future value of expendables, and ownship exposure.
-Ada	25000 lines of Ada code

The two buffers are structured as 2D binary trees indexed on azimuth and elevation angles of arrival. The benefit of organizing trackfiles using binary tree storage lies in the use of corresponding binary search strategies that efficiently locate and sort tracks into 'bins'. Efficient data structures are a consideration since potentially numerous trackfiles will be simultaneously active.

As new datasets are added to the buffers, a distance measure is calculated between each new dataset and the trackfiles already contained in the corresponding buffer. The distance measure is developed from estimated measurement variances. The data is fused to minimize the variance of the combination. The variance estimates are derived using sensor resolution and estimated signal to interference ratios in information theoretic expressions for the particular discrimination procedure. These variances are corrected for estimated sensor loading errors. Tracks judged to be similar are fused into a single Trackfile. This not only improves the quality of the measurements, but helps reduce the number of trackfiles to be stored. If a dataset is not sufficiently similar to any of the existing trackfiles, a new trackfile is created in the appropriate buffer.

Chalkboard Memory. The purpose of the chalkboard memory module is to download data from the sensor postprocessor and convert it to a format recognized by the rest of the CMAT system. The significant feature of the chalkboard memory manager is that it centralizes data storage and allows all system modules to retrieve and update data. This architecture not only provides for efficient storage of all measurement data in one place, but is also flexible to permit future modules to be integrated with minimal interface difficulty. The three primary functions of the chalkboard memory module are: trackfile attribute formatting, all-source attribute fusion, and memory management.

Due to the many sources of data uncertainty[1], multisensor fusion and the associated problem of managing data uncertainty are required characteristics of an automated threat response system. To achieve these characteristics within CMAT, the sensor input data is formatted to be compatible with CMAT's uncertainty management algorithms.

In traditional data fusion systems, Bayesian logic-based techniques are employed to correlate trackfiles and estimate target ID. This method is best suited for discrete-valued inputs, so measured target attributes are represented accordingly. However, not all attribute profiles are conveniently suited for this type of representation. Measurement uncertainty can be oversimplified or ignored, and meaningful parameter details are lost. Consequently, a Bayesian-based data fusion system supports only limited source information, not all-source information, as desired.

Alternately, robust data fusion systems that employ fuzzy and Bayesian based algorithms offer significant advantages over ones that only process discrete-valued parameters. In these robust systems, parameter uncertainties are represented using either mean/variance data (the classical approach) or by fuzzy sets. Fuzzy sets, which are applied to represent continuous-valued trackfile attributes, are a straightforward method for representing all possible values each target attribute may posses across the entire attribute spectrum. Fuzzy sets can be used to describe non-standard, multi-modal target attributions distributions in a rigorous manner.

---

[1]Sources of uncertainty include intelligence information, sensor resolution and interference, sensor overloading, as well as threats that operate using specially reserved wartime operational modes (WARM threats).

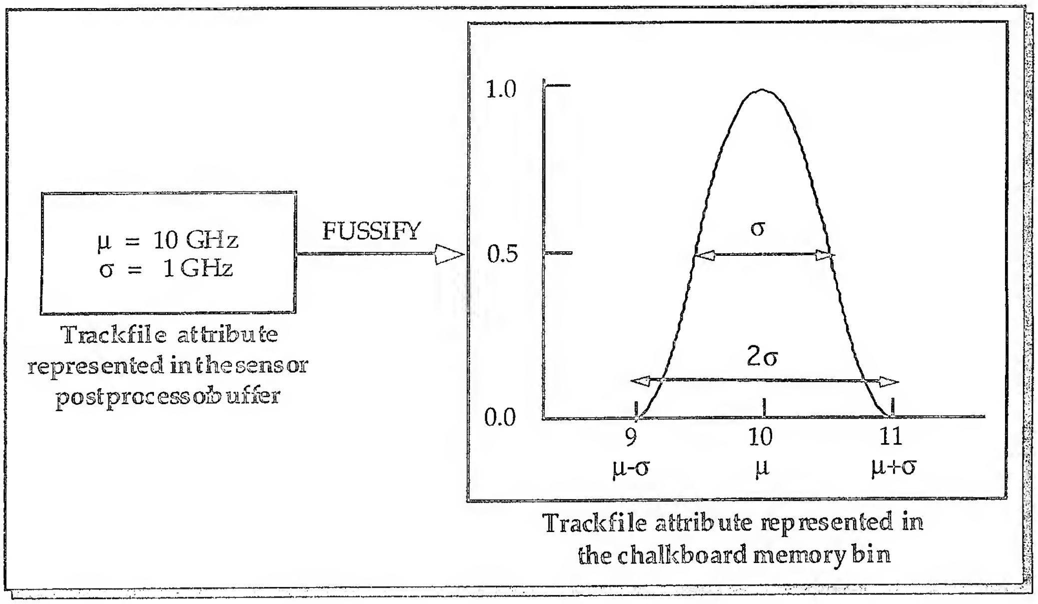


Figure 2. Converting Measurement to Fuzzy Set

Within CMAT, a hybrid combination of both Bayesian logic and Fuzzy logic is used to manage data uncertainty. Each trackfile attribute is represented by using one of these forms[2]. The choice of form is made based on the unique requirements of each attribute. Threat attributes such as velocity, acceleration, and IR color ratios are represented using mean and variance data. Bayesian logic governs how multiple similar-sensor reports are combined for each of these attributes. Threat electronic intelligence (ELINT) data such as RF, PRF, GRF, and PW are represented using membership functions. Fuzzy logic governs how their associated similar-sensor reports are combined. For example, Figure 2 shows how a threats RF signature can readily characterized using fuzzy sets.

The measured feature value converts to the center point while the feature measurement variance converts to a fuzzy set width. Two chalkboard short term memory banks (CSTM), one for missile data and one for signature data are used. After downloading and formatting the datasets from the sensor post processor data buffers, the chalkboard memory module integrates the newly formatted trackfiles with those that are already in the CSTM. This is accomplished by comparing each newly formatted dataset to each existing trackfile in the CSTM using a distance measure. This measure includes a fuzzy set proximity test to compare fuzzy features. Datasets that pass the gate criteria are fused into a single trackfile. Scalar data and fuzzy data are fused to minimize measurement variance. After each newly formatted dataset has been processed the sensor post processor buffers are cleared to make way for new sensor data

As newly formatted datasets are fused with datasets already in the CSTM, a reinforcement coefficient associated with the fused dataset is increased. Conversely, the coefficients associated with datasets that did not get fused are decreased. Thus greater emphasis is given to datasets that are seen repeatedly in successive update intervals. If a particular dataset's coefficient drops below a threshold, the dataset is deleted from memory, reflecting that the particular measurement is no longer confirmed by the sensor systems. The CSTM also stores data from the response option

[2]We are currently exploring adding Shafer-Dempster logic as a third option within CMAT to provide added flexibility with the trackfile uncertainty management options.

generator and threat assessment and prioritization modules about threat classification for use by the rest of the modules in our CMAT system.

Survivability Estimation. The survivability estimation procedure generates a ranked list of appropriate threat response options (countermeasures and/or maneuvers) using the data in short-term memory. This procedure is robust in that it provides optimal countermeasure recommendations even though threats may be operating in wartime modes that may be are different from those catalogued in the threat library. It also makes reasonable recommendations in the presence of uncertain data, or when measurement data is incomplete. Finally, the survivability calculation procedure is entirely database driven so that it does not become obsolete as threats or countermeasures change. Change in these data items is accommodated by updating the corresponding CMAT database.

In the CMAT procedure, three steps determine the effectiveness of the threat response options, Figure 3. In the first step, the signature datasets corresponding to each detected threat are processed through a layer of threat neurons. Each threat neuron calculates a measure of similarity between the observed threat and a threat catalogued in the threat library. In the second step, these similarity measurements are modified based on the threat density probabilities specified in the threat map. In the final step, the similarity data is processed through a layer of countermeasure neurons. Each countermeasure neuron evaluates the effectiveness of a particular response option based on similarity information, engagement geometry, and the countermeasure effectiveness data stored in another database.

The similarity calculation is shown in Figure 4. The Figure shows how the CMAT system handles uncertainty information. A fuzzy pair is formed for each trackfile attribute consisting of: a) the attribute membership function of the observed threat, and b) the attribute membership function of the catalogued threat in the threat library. For every such pair, the fuzzy set intersection is computed. This value is then weighted by the relative importance of the particular feature. These values are summed together to form one value, the similarity measure of the observed threat to the particular known threat.

The survivability estimation procedure makes reasonable countermeasure recommendations when there is incomplete or missing data. For instance, suppose a detected threat is operating in a mode

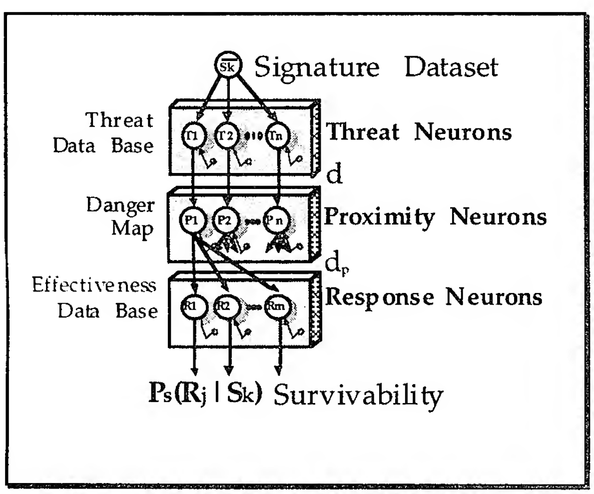


Figure 3 Survivability Estimation

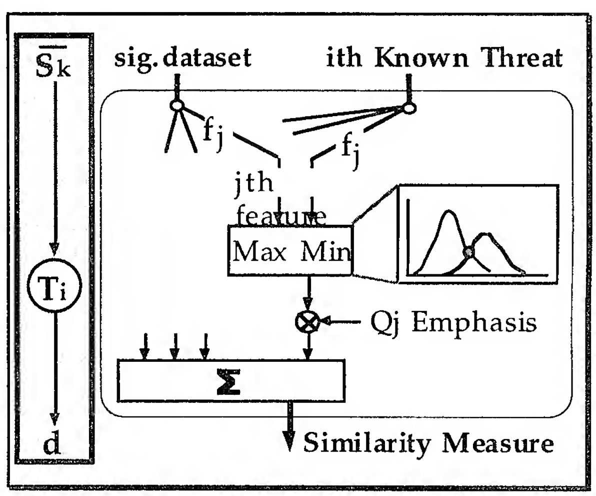


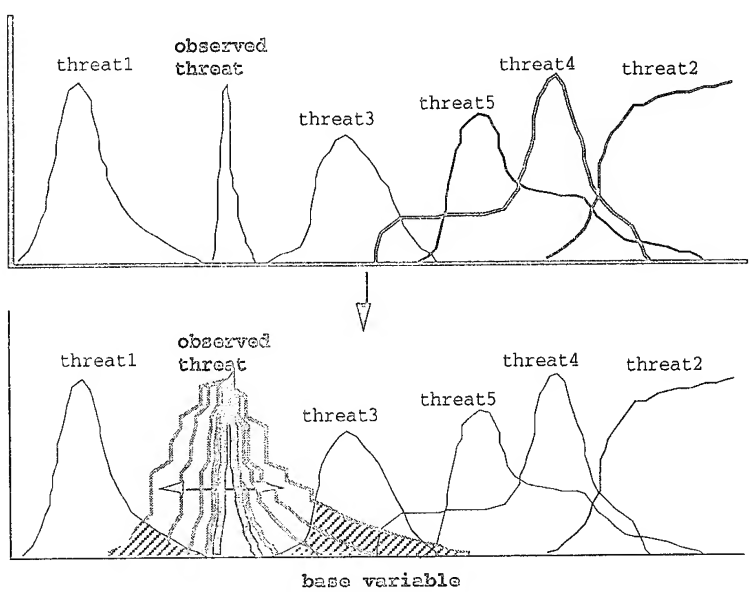
Figure 4. Calculating Similarity Measure

entirely different from those catalogued in the threat library. Such a situation may result in a measured attribute value that has no overlap with the fuzzy set descriptions of the corresponding attribute of the known threats, Figure 5. To mitigate this problem, the fuzzy set corresponding to the particular observed feature is iteratively broadened until some minimum overlap is achieved. The similarity measure for that feature is then recalculated and a response recommendation is made based on this new calculation.

This technique is an implementation of a common-sense generalization algorithm. Simply stated, response recommendations are made that are most effective against all threats that are most similar to the detected threat. When attribute information is lacking and the threat ID process is uncertain, CMAT chooses a robust response that has good effectiveness against all threats that are similar to the detected threat. When there is information that improves the threat ID, CMAT fine tunes its response to optimize ownship survivability. Thus, a robust response is selected when there is limited threat information. A tailored threat response is selected when there is sufficient threat ID confidence to justify the tradeoffs between response robustness (survivability against multiple threats) and response effectiveness (survivability against any one given threat). A summary of these and other key characteristics of the threat neuron procedure is outlined in Table 2.

In the second step of our strategy generator procedure, the similarity measurements from the threat neurons are passed through a layer of proximity neurons. Here, data from the threat map is used. This map is constructed during the premission brief using intelligence information and premission briefing data, which specify the probable number and locations of each catalogued library threat. Finally, each similarity measure is completed by multiplying it with the threat probability estimate. The resulting similarity measure represents the proximity between the observed trackfile and known threats stored in the threat library, with adjustments made for data uncertainty and known threat density probabilities.

Relax feature descriptions as necessary before making final threat similarity decision



THREAT FEATURE

Figure 5. Membership Function Relaxation

Table 2. Threat Neuron Key Features

- Simultaneously accommodates sensor measurement uncertainty and database threat description uncertainty via the use of fuzzy sets.
- Uses a selectable weight to reflect variation in a feature's relative discrimination importance.
- Uses a parallel architecture which allows for efficient processing of the fuzzy pair data
- Uses piecewise analytic functions for membership function representation which allows for very fast closed form analytic solutions to the fuzzy intersection.
- Uses a completely data driven approach.

Figure 6 shows the third and final step in which the similarity measures are passed through a response neuron layer. The survivability estimate is calculated by considering the likelihood of the detected threat being each of the known threats in the database. The countermeasure effectiveness is derived for engagement geometry and kinematic parameters of the engagement. This algorithm has the benefit that its operation does not depend on whether dissimilar signature datasets were fused together in chalkboard memory. This is because the threat response algorithm always considers the likelihood that the detected threat is each known threat when calculating survivability. We again point out that we use a parallel architecture so that processing speed is independent of the number of response options. Also notice that we use a completely data driven approach. Changes in the effectiveness database are made without requiring changes to the response neuron algorithm.

Reaction Selection. The list of generated reaction strategies, ranked by estimated effectiveness, is processed by our Reaction Selection module. This module makes the high level, tactical trade-offs necessary to finalize the strategy selection. The CMAT system considers mission, tactics, complex trade-offs, estimated effectiveness, and aircraft status by directly mimicking off-line databases of expert knowledge. A library of expert knowledge supports this function.

A linear programming based neural network called 'Mimic Net' was developed to handle the unique requirements associated with automating the pilot's cost-benefit tradeoff countermeasure selection intution. The autonomous reaction system was designed to select a response for effective ownship reaction to immediate antiaircraft threats. A list of reaction options along with their estimated survival rates was generated as the set of reaction options to consider. These options were considered together with further trade-offs.

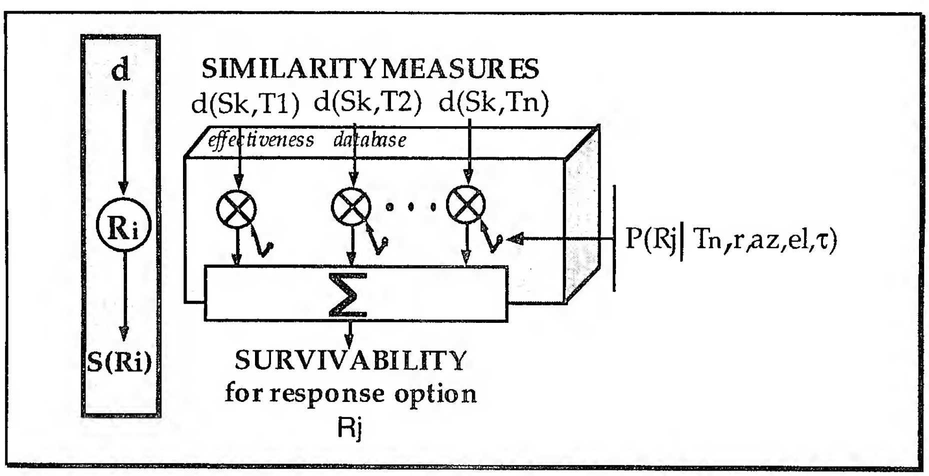


Figure 6. Response Neuron

Several of the difficult to quantify trade-off considerations were the immediate survival risk versus the value of mission success, the potential future value of expendables and fuel, and whether defensive reaction would expose the aircraft to further, deadlier threats. The mimic net approach was conceived as a result of a search for a technique which could encode and generalize from electronic warfare expert and pilot reactions to a wide range of representative threatening scenarios. The training scenarios used in the mimic net training procedure consisted of complete descriptions of the mission and engagement. The expert was given the same information that was to be provided to the automated reaction selection system. Once trained, the mimic net exactly reproduced the expert's threat response reaction strategy.

The mimic net training algorithm utilizes linear programming to discover a feed forward network which, when operated on the training data, exactly mimics all of the selections made by the expert. Training is accomplished by automatically extracting countermeasure ranking constraints derived from each scenario in an off-line training database, Figure 7.

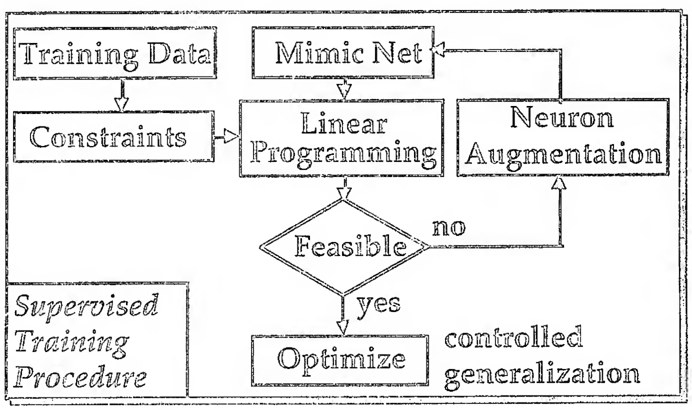


Figure 7. Mimic Net Automated Training Block Diagram

Key Features. Most neural networks require that an expert establish an interconnection architecture for its neural elements before training is initiated. One of the principal features of the mimic net technique is that it automatically discovers an architecture which is best suited to the training data. The construction procedure produces the smallest size network that accommodates the training data. As a result, the mimic net procedure discovers a network architecture that possesses the greatest degree of generalization. Furthermore, the analyst is provided flexibility for exercising precise control over how the network extrapolates from learned behavior to new problem situations.

Databases. Integral to the CMAT threat response recommendation system are the databases containing the descriptions of the known threat systems and countermeasure survivability profiles. The first such database contains all threat attributes described in terms of fuzzy sets, such as radar frequency, pulse repetition frequency, and pulse width. A second database contains descriptions of the threat inner and outer launch envelopes. A third database contains the compressed survivability profiles.

Data compression for the problem of automated threat response recommendation is essential to allow real-time integration of the multidimensional survivability profiles. CMAT employs a data compression scheme that is based on physics-derived separable functions in conjunction with

analytic functional fits to the numeric aspect-dependent survivability data. The result is a highly compressed analytic representation of survivability, which enables survivability profiles to be integrated in real-time across all dimensions of survivability dependence. The limits of integration are established based on the target engagement geometry estimates and the associated estimate uncertainties. Thus, survivability is estimated as

$$\int_{\mu-\sigma}^{\mu+\sigma} S \, dS$$

where

μ = mean vector of survivability dependence factors
= [azimuth, elevation, range, time-to-go, altitude, and air velocity]
σ = standard deviation associated with **x**
S = survivability as a function of **x** and σ

One recent technical advancement enabling the efficient compression of large tables of reference data for use in an automated system is Chebyshev functional expansion of multidimensional tables. This methodology simultaneously achieves three goals:

- Large tables or figures of data are compressed into a limited number of expansion coefficients
- The fit process is a well conditioned, efficient numerical procedure permitting an interactive graphical interface
- The average value of the function, necessary to assessing the mean value of the data over uncertainty regions, is nearly as readily evaluated as the function itself

The Chebyshev polynomial fit is closer in method to discrete Fourier expansions than to least squared (LS) error polynomial fits, although the fit is in terms of polynomials. Instead of minimizing the fit error at each of the input data points, this method uses a local polynomial fit to evaluate the interpolated function at preselected points, and then passes a polynomial curve through those points.

In a LS fit, the order is determined and then the polynomial of this order with the least squared error, summed over all the input data points, is evaluated. This evaluation involves the inversion of the linear, normal regression equations. This inversion is a notoriously ill-conditioned problem. And even when the inversion is numerically feasible without resorting to a singular value decomposition, the inversion is much less efficient than the FFT based algorithms we have employed.

It has been observed that the Chebyshev polynomial is approximately equal to the minimum maximum error (min-max) polynomial fit, as opposed to a LS fit. The advantages of this approach are:

- Reduced operation count:
  $NM(\log2(N) + \log2(M))$ versus $(NM)^3$
- Fit is well-behaved numerically versus ill conditioned matrix inversion
- Approximate min-max rather than LS fit criterion

The purposeful selection of a fit criterion provides for an efficient evaluation of Chebyshev polynomial expansion coefficients. The Chebyshev polynomial expansion coefficients, c_k, are defined by:

$$f(x) \approx \sum_{k=1}^{N} c_k T_{k-1}(x) - \frac{1}{2} c_1$$

The choice of expansion and the discrete orthogonality of the Chebyshev polynomials provides a simple inversion formula for the expansion coefficients:

$$c_j = \frac{2}{N} \sum_{k=1}^{N} f(x_{kN}) \cos(\pi(j - 1)(k - \tfrac{1}{2})/N))$$

Thus, the evaluation of the coefficients requires an orthogonal, cosine transform of function values at selected points. This evaluation is performed efficiently using Fast Fourier transform methods.

From the Chebyshev polynomial expansion of a function, a Chebyshev polynomial expansion for the integral of the function is readily derived. Thus, averages of the fit function are evaluated nearly as efficiently as the function itself.

Since our concern is to maximize the survival chances of ownship, the probabilities of survival in the range 0.5-1 are of more concern than small probabilities. Data fitting errors lead to estimated effectiveness values greater than one or less than zero. Both concerns are mitigated by performing the curve fit to a function of the actual effectiveness. The desired function has a conveniently calculated inverse function, forces the fit fidelity to be best at higher effectiveness values, and limits the effectiveness to less than unity. All these desirable features are achieved by the choice:

$$f(x) \equiv 1 - (1 - P(x))^a$$

The model curve fit is made to this function, rather than to the actual effectiveness values, $P(x)$. If any calculated value for f is less than zero, it is truncated to zero with no significant loss of fidelity since effectiveness values near zero are of no interest.

Conclusion.

This paper described the CMAT methodology for automated countermeasure response recommendation. The various components of CMAT were described, including the sensor post processor, chalkboard memory, survivability estimation, and response optimization modules. The CMAT design emphasizes the importance of a data driven architecture, the careful handling of threat and intelligence uncertainty, and the need to compress the numeric databases so that numeric procedures can be used instead of generalized rule bases when calculating survivability. CMAT has passed the feasibility demonstration phase and is now being integrated within the SH-60B platform for its debut application. The CMAT feasibility development was sponsored by the Air Force under a Phase I/II SBIR effort. The current SBIR phase II SH-60B integration effort is sponsored by NAVAIR.

COMPUTER ASSISTED TEST DEVELOPMENT AND REPORTING SYSTEM (CATDARS)

Douglas C. Cook*
Mahlon R. Haunschild
Thomas L. Rossi

Sverdrup Technology, Inc./TEAS Group
4200 Col. Glenn Hwy.
Beavercreek Ohio, 45431

BACKGROUND

In recent years we have seen remarkable advances in digital technology that have enabled our weapons systems to achieve unparalleled advances against various threat systems that exist in the world today. The direct by-products of these advances are that we have developed systems that rely more and more on software to adapt to changing and evolving threats rather than hardware. For every new or changing threat that we wish to address with overall system capability, we have to make a change in the Operational Flight Program (OFP), or the software, that controls how the hardware analyzes the data observed or collected in a war-time operating environment. Unfortunately these "simple" software changes are never as straight forward as we might hope. In other words for every change in one source line of code that we wish to implement, to address a new requirement for a US. weapon system, we absolutely must ensure that the changes that have been introduced did not in any way adversely effect the overall weapon system level performance against all other requirements. This system level testing occurs at various contractor and Department of Defense (DOD) facilities around the country depending on the system undergoing testing. Since the test methodology used is location specific depending upon what Radio Frequency (RF) generating hardware is available to emulate the threat, and since the information that describes the threat characteristics may be contained in multiple databases, the task of developing sound test procedures for Radar Warning Receiver (RWR) OFP testing is a very complex matter. The system level verification testing that is required for updated RWR OFPs is therefore highly dependent upon the skills of the personnel performing the tests and their site specific test methodologies that have been developed over the years. The Computer Assisted Test Development and Reporting System (CATDARS) program was initiated to assist the test engineers in developing the test scenarios required to test the updated OFPs. This paper discusses and gives an outline of the CATDARS program and addresses in particular the RWR system level testing that is performed in the Electronic Warfare Integrated Support Facility (EWAISF) of the Simulation and Analysis Group (WR/LNEV) of the Electronic Warfare Division (WR/LN) at Warner-Robins Air Logistics Center (WR-ALC), Robins AFB, GA.

Overview

One component of Warner Robins' Simulation and Analysis Group is to test OFP updates to the RWRs that they have sustaining engineering responsibility to maintain. Currently, this process is very labor intensive and relies heavily on the skilled engineers and technicians at WR/LNEV. To improve the process of testing the RWR OFPs, the Wright Laboratory's Readiness Technology Branch (WL/AAAF) in concert with WR/LNEV have initiated several programs to improve the overall testing efficiency of WR/LNEV and therefore reduce the overall 18-month cycle for an OFP update. These projects include A Digital Avionics Methodology Schema (ADAMS), Low Cost Interactive Stimuli-Generating Test Station (LISTS) and CATDARS. ADAMS will develop, prototype, and demonstrate a methodology to perform more rigorous, timely, at RF testing of EC system/subsystem OFP software. LISTS is developing a low-cost, portable, high fidelity microwave threat environment for stimulating the EC equipment while the concept behind the CATDARS project is to derive and demonstrate a generic software based tool that utilizes current testing knowledge to aid in the development of emitter test files that drive the RF generators used in the testing of RWR OFPs.

This paper describes the efforts underway by Sverdrup Technology Inc./Technical, Engineering and Acquisition Support (TEAS) Group under contract with the US Air Force to develop a prototype CATDARS system. The CATDARS program objectives are to formulate and demonstrate a generic software based tool that acquires and exploits current EC system testing knowledge relative to the WR/LNEV testing of RWR OFPs. The goal is to automate the front end functions required to define and create the required threat definition test case files for EC system testing. The CATDARS program will develop a potential software tool for extracting information from the Electronic Warfare Integrated Reprogramming Database (EWIRDB). The information obtained will support the generation of emitter test files that are used to drive an RF generator, The Advanced Standard Threat Generator (ASTG) that injects an RF signal to excite an RWR to test and evaluate the performance of new RWR OFP releases. Additionally CATDARS will provide for a files management system to update and maintain test files created using the CATDARS system. This effort will also potentially provide a core software tool that is easily customized by other individual tasks to provide a desired unique output. Most of the software tools in use today that utilize the EWIRDB are very good at what they were designed to accomplish, however, most were tailored to a very specific application. Our goal is to develop CATDARS to have an inherent modularity so that it may be used in a variety of applications other than RWR OFP testing (i.e. building Mission Data Files for weapons or electronic jammer pods).

At the heart of the CATDARS effort is capturing the processes that are used to develop a new test file for a particular threat. Our goal is to accomplish this utilizing new and evolving commercially available database software that provides the opportunity to improve and automate the use of the EWIRDB and other electronic parameter databases. The key is to be able to access the EWIRDB information to allow for a much more user friendly operating environment in which to build the needed test files. The rapid

development of many new, widely used, and relatively powerful database systems provides the opportunity to develop more flexible tools and more easily tailored queries of the EWIRDB for each individual EC system. Moreover, while working with the EWIRDB, these tools may be used on a personal computer (PC) allowing interaction with other PC software. This paper explores the application of such popular database software to the standard EWIRDB products provided by the National Air Intelligence Center (NAIC) for the generation of valid threat test files.

The CATDARS methodology is scheduled to be demonstrated in March 1996 using an AN/ALR-69 Mod IV RWR stimulated by an AN/ALM-234 (the ASTG) EC test-bed located at Robins AFB, GA. The ALR-69 Mod IV was selected because it is currently undergoing testing at WR/LNEV and seemed the most viable test platform. The ALR-69 is the RWR that is currently used onboard the F-16 Falcon. For the demonstration at the end of this phase of the CATDARS program we will only be working with a subset of approximately 20 Electronic Intelligence (ELINT) Notations (ELNOTS) out of the entire EWIRDB.

SYSTEM DESCRIPTIONS

The CATDARS system may be required to operate with several other systems that are currently being developed for use in the EWAISF in the conduct of RWR test activities (see figure 1). Short descriptions of these systems are provided in the following paragraphs so that the reader can better understand the overall system architecture and division of functional responsibilities associated with the test activities that this system is envisioned to be part of in the EWAISF. The collateral systems described in the following paragraphs include the ASTG, ADAMS, and LISTS.

ASTG

The ASTG is an electronic warfare threat simulator system that is used for the testing and validation of airborne RWR and Electronic Countermeasures (ECM) at WR/LNEV. It consists of an RF threat signal generator unit, a VAX 8600 computer system, controlling software programs, VAX computer system software, and various terminals, tape drives, printers, cables, etc.

The ASTG generates RF signals that simulate electronic threats in a real-world environment. These signals are introduced to the antenna connector(s) of the unit under test, and the unit's performance is then measured to determine its behavior.

ADAMS

The ASTG test methodology is an example of a manually-operated test process because the operator must manually configure the ASTG to generate a threat signal and then note the test results on measurement instrumentation. ADAMS automates the test loop by commanding the ASTG Scenario Execution program to

generate sequences of threats as described by the ASTG's scenario and emitter files, and then notes the test results from the unit under test.

ADAMS is a PC-based system that controls the ASTG through its existing forms-based video terminal interface, and collects test data from the unit under test via a MIL-STD-1553 data bus interface. ADAMS provides a script file capability so that the user can automate the testing process for a number of threat signal environments and then record the test data for each threat. An initial version of ADAMS, known as ADAMS I, was demonstrated at WR/LNEV with the ASTG and an ALR-69 RWR. During this demonstration the ADAMS achieved a 50% reduction in test time.

LISTS

The LISTS program is intended to develop a low-cost, portable, high fidelity microwave threat environment for stimulating the EC equipment. It will consist of a control computer system and an RF generator chassis that will provide enhanced RF threat generation capabilities.

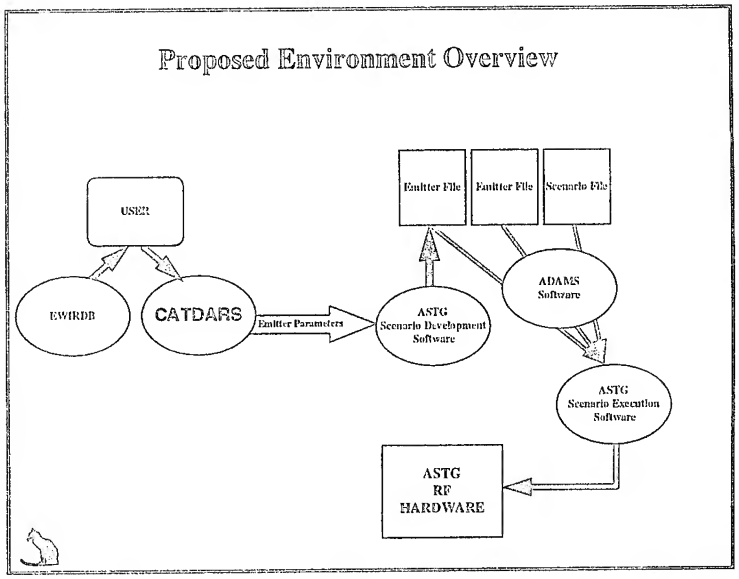


Figure 1. Environment for CATDARS

CATDARS REQUIREMENTS

The ASTG emitter file development process requires the analyst to examine the EWIRDB for candidate emitters, select valid sets of data for the emitters' operational modes that are of interest, adjust this data to reflect other sources of information or specific test requirements, and enter this data into the ASTG using the ASTG's Scenario Development program. The Scenario Development program then produces the emitter and scenario files that the ASTG uses to simulate a threat environment. The generation and maintenance of the ASTG's emitter files is a particularly laborious process because the analysts who do this work must manually browse the EWIR database, as well as other sources of information, to find their information. Also, the information in the existing emitter files must be compared against new assessments of those emitters in the EWIRDB updates to keep the emitter file characteristics up to date.

CATDARS will ease the emitter file generation process by automating the low-level mechanics of the EWIRDB browsing and file maintenance processes. The analyst will then be able to spend more time in determining the required characteristics and parameters of test emitters, instead of becoming bogged down in the emitter data collection process.

The CATDARS system is intended to provide the analyst with on-line access to the EWIRDB. The analyst will use CATDARS to select an emitter from the EWIRDB that would be used in a particular test, using a Graphical User Interface (GUI) based point-and-click user interface presentation. The analyst will then be able to select the ELINT parameters for his emitters from the database and use CATDARS' point-and-click user interface to call up supporting information from the database for particular characteristics, including suffix codes, suffix tables, references, and comments. The analyst will be able to modify any or all of an emitter's parameters based on the information that he derives from his analysis of the emitter's characteristics. Once the analysis process is completed, the analyst will then save the emitter and it's parameters into a test file for later use with the ASTG. CATDARS will also provide an interface to the ASTG Scenario Generation program to build the ASTG emitter files, using the information contained in the CATDARS file management system.

INTERACTIVE TEST FILE BUILDING

In order to build a test file, all the information needed to characterize the emitter must be captured. The user obviously cannot be the sole source of all of this information. Nor could CATDARS anticipate the personality of the test for which the file is being built. Therefore, the user and CATDARS must build the test file in an interactive environment. CATDARS must be able to provide the user access to various sources of information depending on user preferences, type of test, etc... while relying on the user to provide guidance. Examples of information that CATDARS will be required to allow user-access to include:

- Parameter values from previous test files; this will provide insight on parametric values chosen for previous tests.
- Parameter values from the EWIRDB; to provide the range of values available for a parameter, as well as, the latest intelligence data.
- Similar parameters for similar emitters from the EWIRDB; to provide intelligence data for emitters that may share performance characteristics that are similar.

CATDARS will walk the user through various screens while extracting the information needed. The information will be recorded into the CATDARS file management system for use in building the ASTG test file.

FILE MANAGEMENT

Test files may be generated from scratch, re-used, modified slightly for a related test, or simply referenced for individual parameter values. For example, if the

user wishes to build several very similar test files, the file management system will provide access to values used in previous files. This will allow the user to simply modify the one or two parameter values that have changed between the tests.

Out-dated test files may contain parameter values extracted from the EWIRDB that have now been updated. If this now out-dated test file is referenced, it is desirable that CATDARS provide the user some feedback. The problem lies in the ambiguous relationship between the parameter value, its true value, and the multiple values maintained in the database. The following examples illustrate the complexity of attempting to recognize EWIRDB dependent, out-dated parameter values:

- A parameter value may have been modified due to ASTG limitations
- A parameter value may have been chosen solely on the nature of the test
- A parameter value may have multiple values in the EWIRDB due to various sources or different modes of operation, each with different dates.

For these reasons, CATDARS may provide EWIRDB comparisons, but not an automated update procedure.

CATDARS Interfaces

CATDARS will be required to have interfaces with the user, the EWIRDB, and the ASTG. Figure 1 illustrates the relationships pictorially. The user in conjunction with the EWIRDB provides the parametric data required to build the emitter files. The ASTG then uses these emitter files to simulate the threat.

The user interface is the most important interface for the CATDARS project. In order for CATDARS to be successful the human interface environment has to be more user-friendly than conventional approaches. It needs to provide the user with a graphical interface allowing multiple windows with various information. The environment must be customizable depending on the user's experience and the application of the test file. For example, a user with considerably less experience may require many more windows containing comments, references, definitions, etc.. while an experienced user can assimilate the information with less detail. Also, an emitter file may be generated with parameters completely defined by the test requirements and would therefore not require any information from the EWIRDB.

The EWIRDB will provide the parameter values necessary to generate the test files. EWIRDB comes in several different formats and the exact format required for the CATDARS interface has not been determined. Currently, a relational database is being considered for the underlying engine of CATDARS. For this reason, the EWIRDB has been ordered in two formats favorable for relational databases (DBMS Flat File, and ORACLE). In addition the TERF format, most commonly used, is also being considered. Additionally, CATDARS may be

required to interface with the EWIRDB CD-ROM Query System and Suffix Matrix Expander. Both products have been developed by NAIC with years of experience and user feedback. For this reason, they may be the best browse applications to meet the requirements of CATDARS.

Ultimately, CATDARS must interface with the ASTG to provide the parametric data captured during the test file generation. The format of the ASTG input file is complex and the methodology of the software to generate this file is not readily duplicated. The ASTG software has very limited interface capabilities and therefore the current approach to interfacing with the ASTG involves two steps. The first step will be to generate a Test File Script that contains all the keystrokes necessary for the ASTG Software to build the Test File. The second step will be emulating the keystrokes contained in the Test File Scripts that will build the emitter file in the ASTG Software.

CATDARS Technical Approach

Sverdrup Technology has developed a technical approach for the development of the CATDARS concept demonstration system. This approach is outlined in the following paragraphs.

1. Sverdrup first determined a candidate set of requirements for CATDARS. This set of requirements contained the CATDARS top-level system and interface definitions, functional requirements, and design constraints. These requirements served as a structural starting point for further research and investigations.

2. Sverdrup then investigated existing EWIRDB tools and test processes to determine their applicability to CATDARS. The EWIR tools discussed below all mechanize the EWIR browse/select function and the suffix matrix expansion function to some degree. It is to our customer's advantage for us to examine these tools to better understand these processes. It may even be possible to use some of these tools as a part of the CATDARS system, instead of re-inventing an existing tool. It is also critical that we understand the analysts' actions and thought processes so that we can refine functionality and the user interface. To date, we have examined the following tools and applications:

   Computer-Aided Electronic Warfare Information System (CAEWIS): This tool is used by the USAF Air Warfare Center at Eglin AFB for browsing the EWIRDB and expanding the suffix matrix. It runs on Sun workstations and uses the Sun X-Windows graphical user interface. The CAEWIS user can select an emitter of interest by ELNOT and then examine that emitter's parameters. If desired, the user can also expand the emitter's suffix matrix and then examine the characteristics of the emitter's mode combinations. CAEWIS can produce limited user specified output files of extracted or modified data.

EWIRDB CD-ROM Query System and Suffix Matrix Expander: NAIC distributes an EWIRDB query program and suffix matrix expander program with the EWIRDB on CD-ROM. These programs provide the end user of the EWIRDB with the capability of browsing the EWIRDB and extracting emitter information into disk files for further use. These applications are IBM PC-based as well as UNIX-based and use a graphical user interface.

WR/LNEV Tools (BROWSE, EXPAND, TAP): These and other applications were developed over the years by personnel at WR/LNEV for browsing the EWIRDB and expanding the suffix matrices. The tools run on DEC VAX computers under the VMS operating system.

The CD-ROM query system and suffix matrix expander showed the most promise as applications that could be linked into the CATDARS software system. This could save us from developing these applications from scratch and allow us to concentrate on the EWIR analysis support and file manipulation capabilities of CATDARS.

3. Next we looked at and began analyzing the ASTG and its interfaces. This work when completed will allow us to develop an effective CATDARS - ASTG interface. Our understanding so far includes the following:

The operator programs the ASTG to simulate the desired threat emitter by using an ASTG VAX software application known as the Scenario Generation program. The operator uses this program to construct a model of the particular test scenario and associated emitter that is required for a particular test. The Scenario Generation program's output is stored in two VAX disk files, known as the scenario description file and emitter description file. The scenario description file contains the information that the ASTG RF generator uses to simulate emitter locations, distances, etc. while the emitter description file contains the information that the ASTG RF generator uses to simulate the threat emitter's RF frequency, PRI, PW, and antenna pattern and scanning characteristics. The operator must use the Scenario Generation program to build a scenario file and an emitter file for every combination of emitter modes and scenarios that he needs to simulate. CATDARS must provide an interface to the ASTG Scenario Generation program in order to transfer the emitter information from CATDARS into the ASTG emitter files

4. The next step in our overall approach is to refine the requirements from step 1 based on the information captured in steps 2 and 3 to produce a detailed set of CATDARS system requirements and final system architecture. This will be used to produce a CATDARS prototype that will operate with the ASTG to produce actual test files for a limited number of candidate EWIR emitters.

5. At the conclusion of the CATDARS demonstration phase of the program the prototype CATDARS system will be transported to WR/LNEV for a functional demonstration of its capabilities with the ASTG in a real testing environment. The CATDARS host computer

is anticipated to be connected to the ASTG through an existing RS-232 serial terminal line so that it can communicate with the ASTG. Once connected, CATDARS will be tested both by itself and in conjunction with the ASTG to confirm that it meets requirements, and also to validate the CATDARS user interface with the user community.

POTENTIAL OTHER USES FOR CATDARS

One of the potential pay-offs with the CATDARS program is that the concepts employed here should be able to be used by other agencies who draw on information contained in the EWIRDB or similar intelligence databases for the generation of test files or other sets of information as may be required by their particular endeavor. We are attempting to develop the CATDARS system such that it's underlying structure and design methodology may be tailored to support other systems/agencies such as the EWOLS and STEMS at Robins AFB, GA, the CEESIM and the Electronic Warfare Operational Test System (EWOTS) at the 513th ETS at Offut AFB, NE, the Air Warfare Center and the Advanced Multiple Environment Simulator (AMES) at Eglin AFB, FL, NAIC at WPAFB, OH, and the 412th Test Squadron at Edwards AFB, CA.

CONCLUSIONS

The CATDARS program is underway with a large portion of the requirements defined and specific implementations being worked at this time. We are currently on schedule to demonstrate in March 1996 the CATDARS concept using the ALR-69, Mod IV as the test unit in conjunction with the ASTG at WR/LNEV. As of the writing of this paper indications are that we'll be able to meet the project objectives of hosting this software tool on a PC based system and provide a level of automated test file building capability that is currently not available.

PC-BASED RADAR ENVIRONMENT SIMULATOR

Raymond L. Durand
Technology Service Corporation, Santa Monica, CA 90405

Introduction

Radar Environment Simulators (RES) have proven to be very useful in the development, testing, and operator training of radar systems. A RES allows the radar to be used in a laboratory environment without requiring a cooperative target environment. It can inject simulated signals directly into the radar's receiver or can be configured to radiate its output toward the radar under test in an anechoic chamber or test range. The operator can define a simulated scenario which typically includes targets (aircraft, missiles, etc.), clutter (land, sea, weather, etc.) and even jammers which test the radar's ability to operate in the presence of electronic countermeasures. These scenarios can be run repeatedly to evaluate radar performance, trouble shoot problems or to train operators.

This paper describes a low cost, PC-based RES developed under a company-sponsored IR&D program, that maximizes the use of commercially available off-the-shelf (COTS) hardware and software. The development effort benefits from more than twenty years of company experience in providing RES solutions for a wide variety of radar simulation requirements.

The RES includes high speed digital signal processing (DSP) modules housed within the PC that perform real time signal generation of target and clutter signal returns. A Virtual Instrument Control Panel, based on National Instrument's *LabVIEW® for Windows*, provides a flexible, industry standard graphical user interface that facilitates operator training and provides a mechanism for custom test configurations.

Technical Overview

A RES generates a real-world simulation of the environment that a radar would see if it were radiating in its normal operating mode. The RES must be fully synchronized with the radar and have access to the radar's timing and control signals. These signals include the radar's Pulse Repetition Frequency (PRF), antenna pointing angles (azimuth & elevation), transmit frequency, and range clock signals. Radar synchronization represents a real-time challenge to a RES, particularly with modern electronically scanned antennas, since the RES must be able to respond to nearly instantaneous changes in the radar's timing and antenna parameters.

Traditional RES systems utilize a host computer that works with custom designed hardware to provide real time injection of radar environment signals into the radar. The host contains the data base of parameters used to generate radar environment scenarios and one of its principal functions is to channel this data to special purpose, real-time signal generating hardware. This channel often becomes the bottleneck in RES systems, since random access to the large volume of radar environment data is limited by disk/tape access times and computer bus bandwidths.

The complex timing and interface requirements of a RES make it a very specialized piece of test equipment which is often priced beyond budgets of test and training organizations. In response to demands for lower cost RES solutions, a new RES has been developed which is based on an IBM compatible personal

computer and modern DSP technology. The new approach not only results in a lower cost system, but one that is flexible, easy to use, and requires less physical space than previously available systems.

The following block diagram and technical description further describe the major components of a single channel RES. Multiple channel systems, i.e., a RES configured for a monopulse radar, can be configured with additional plug-in modules described below.

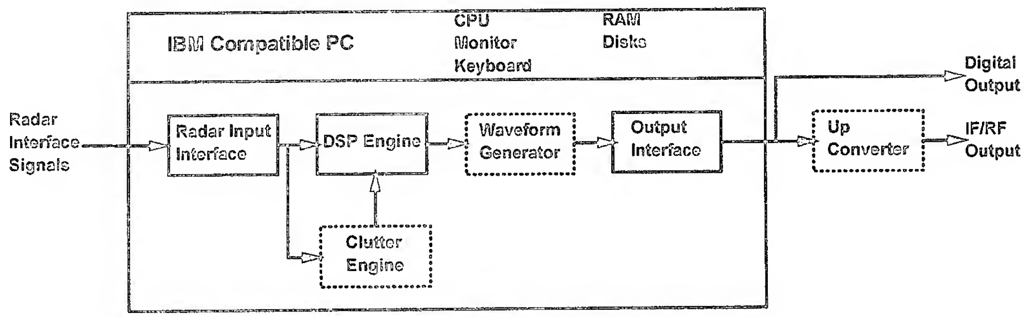


Figure 1. Single Channel RES Block Diagram

IBM Compatible PC

The RES digital electronics are housed within an IBM compatible PC. The PC provides the chassis and power supply for the digital plug in modules. These modules provide the interface with the radar and generate the simulated radar returns. The PC itself is not involved in real time processing; it provides the operator interface and supervisory control of the digital modules. The speed and performance capabilities of the PC are not factors in RES signal generation. The performance of the PC is merely reflected in the responsiveness of the operator interface, which is described later in this paper.

Radar Input Interface

The Radar Input Interface module obtains the synchronization signals from the radar and communicates these signals to the remaining digital modules within the PC. The signals include the radar's PRF, range clock, azimuth and elevation angles, transmit frequency and other pertinent control information. The module contains an area for user defined, custom circuits which provide the bridge between the radar's unique signal interface definition and the interface used internally to the RES.

DSP Engine

The Texas Instruments TMS320C30 floating point processor serves as the heart of the signal processing used in the RES. Operating at 40 MHZ, the device is capable of performing 32 bit, floating-point operations at a rate of up to 40 MFLOPS (millions of floating point operations per second).

In a major departure from conventional RES designs, high density, static random access memory (SRAM) is used to store scenario information within each engine. This SRAM is initialized by the PC during system initialization, and contains information for the entire scenario. This eliminates the need for real-time updates by the PC, and simplifies the system design. This SRAM is also *dual-access* - both the PC and the DSP have uncontested simultaneous access to data and control parameters.

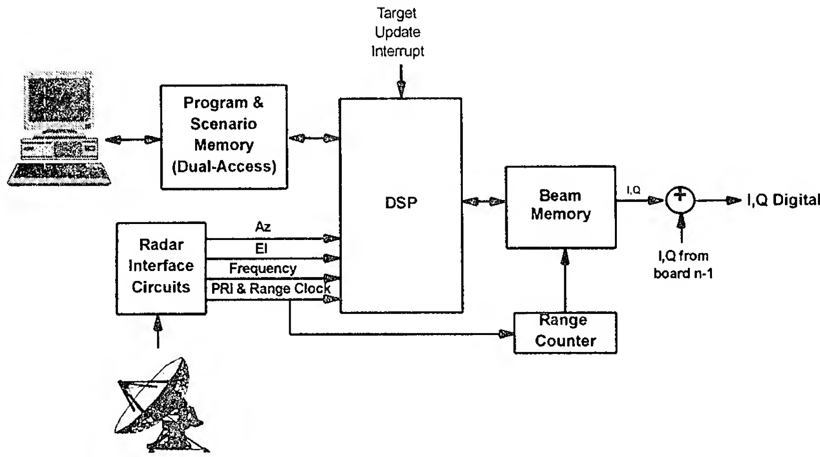


Figure 2. DSP RES Engine Block Diagram

The DSP Engine is synchronized to the radar via the radar interface circuitry, which provides the basic timing and control information from the radar. Radar environment information is stored in the SRAM in a time ordered list of update parameters, from which the DSP performs the calculations to provide targets and clutter in response to the radar's current operating mode and field of view. The DSP indexes through the scenario, typically with a resolution on the order of 10 to 100 Hz, using a programmable, timed interrupt.

During each pulse repetition interval, the DSP must determine which signal returns are within the radar's field of view and perform the calculations to accurately represent these returns with respect to the radar's PRF, transmit frequency, and antenna weighting. Firmware, which is loaded by the PC during initialization, executes assembly language routines which operate in a real time, interrupt driven environment to perform the required calculations. A dual port memory is used to store the calculated 16 bit in-phase and quadrature (I, Q) signals which are read out at the radar's range clock rate. Overlapping signals are accommodated by performing read-modify-write cycles which sum the returns in overlapping range cells. A final adder is provided to sum the results of additional modules which may be added to the system to increase signal density, i.e., additional targets, clutter, etc.

Clutter Generator Engine

The optional Clutter Generator Engine is a dedicated clutter processor which provides a high density clutter environment for system requirements beyond the capability of the standard DSP Engine. It consists of a two-board design based on the TMS320 digital signal processor and can be configured to output three monopulse channels.

In both the DSP Engine and Clutter Generator modules, clutter is created by using computer generated spectral tables that are accessed in a manner which simulates the distribution, bandwidth, and velocity offset of the desired clutter simulation. Clutter capabilities include the generation of various forms of clutter including sea, sea/land boundaries, birds, weather, and other natural physical phenomena.

Waveform Generator

The Waveform Generator is used to match the RES output to the pulse compression used in the radar. Its function is to convolve the compressed pulse signals generated by the DSP Engine with the radar's waveform.

The convolution process is performed digitally in the time domain, using digital filter circuits. The complex filter coefficients are computed off-line by digitizing the radar waveform, and can be quickly changed to follow the radar's pulse mode changes. The basic Waveform Generator provides a 256-point radar waveform. The number of points is calculated by the time-bandwidth product of the radar waveform, i.e., a single Waveform Generator is capable of simulating a 25.6 usec coded pulse with a bandwidth of 10 MHZ.

The input precision of the Waveform Generator accommodates the 16 bit I, Q signals generated by the DSP Engine. Coefficient size is 7 bits each, I and Q, and is programmed via the PC bus interface. The Waveform Generator maintains full dynamic resolution throughout all filter computations. A gain control is provided to scale each of the resulting 32 bit I and Q products into 16 bit resolution for further processing. The Waveform Generator is modular, and can be expanded both in length (to accommodate longer pulse widths) and in the number of coefficient bits used to represent the waveform.

Output Interface Module

The Output Interface Module accepts data from the DSP Engine or Waveform Generator. It has space for user defined circuitry which may be necessary to interface analog or digital video with the radar and includes circuits to provide a feedback path to the PC for self test purposes. It also provides the physical interface with the Digital to IF/RF Up Conversion Assembly. Digital to analog converters are provided to monitor the final I and Q outputs for test purposes.

IF/RF Up-Converter Assembly

The IF/RF Up-Converter Assembly is a separately packaged unit which converts the digital I and Q signals from the PC chassis into IF or RF outputs for injection into the radar. The design is implemented using direct digital synthesis which minimizes offset, balance and image generation problems characteristic of analog designs.

Software

The RES operates under a standard Microsoft Windows™ environment. A Windows application developed using *LabVIEW for Windows* (National Instruments) provides the operator interface and control mechanism for the RES. A Virtual Instrument Control Panel using the PC's display, mouse and keyboard allow the operator to control and monitor system performance.

The PC is not involved in any real-time processing. The Windows application functions independently of the signal generation performed by the DSP plug-in modules. The application communicates with the DSP modules through dual-access memory which is shared between the DSP modules and the PC. This common memory serves as a convenient interface to communicate control and status information between the two processes.

All real-time processing is performed by the various custom designed plug-in modules. These modules are based on the TMS320 DSP and are optimized to efficiently execute various target and clutter signal generation algorithms. Software for these processors is written in assembly language, and is downloaded into the DSP modules during initialization by the PC.

Operator Control Panel

The Virtual Instrument Control Panel allows the operator to configure what types of signals are generated and injected into the radar. Individual controls are provided to specify target velocity, radar cross section, azimuth, elevation and range from the radar. Radar waveform modulation, (e.g., Barker Code, Linear FM, etc.) is programmable by the operator. Clutter controls allow the operator to specify various types of clutter and specify parameters such as gain, bandwidth and velocity offset. The operator can elect to run a preprogramed scenario of targets and clutter which is defined by the operator in an off-line, *LabVIEW for Windows* application.

In addition to controls, the Virtual Instrument Control Panel provides a visual feedback of target dynamics with its Plan Position Indicator as well as individual plots of target parameters.

A built in test capability is included to monitor the quality of signals generated by the RES using the system's Fast Fourier Transform (FFT) display. Controls are provided to specify test parameters, including the size of the FFT. Using the built in test features, the operator cannot only verify the operation of the RES, but can also compare the characteristics of the simulated signals with the observed results of the radar under test.

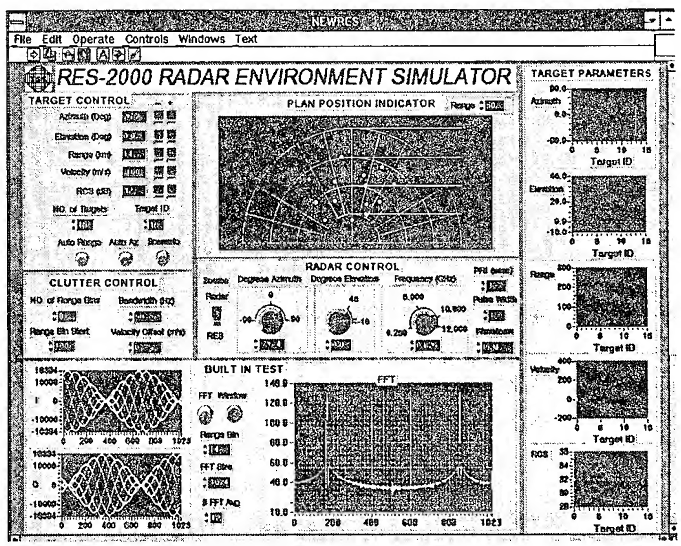


Figure 3. Operator Control Panel

User Configurable Parameters

Older generation RES systems typically required RES and radar characteristics to be hardwired or were contained in firmware which were inaccessible to the user. The RES is designed to provide convenient access to most radar and RES parameters. Standard ASCII text files contain most radar and RES parameters which are used during the course of initialization and real time signal generation.

Examples of user defined parameters include antenna pattern tables that model the azimuth and elevation profiles of the radar's antenna, and the radar waveform. In both cases, the user can modify or create new ASCII text files for various custom designed RES configurations.

RES Benchmarks

The performance of the DSP Engine is limited to the number of calculations that can be performed during the pulse repetition interval of the radar. For high PRF radars, or for systems where a high density of targets is required, multiple DSP modules can be configured to meet the desired scenario loading. For

benchmark purposes, the current RES Engine is capable of executing all of the calculations required to generate a single target, within the radar beam, in approximately three micro seconds. For a one KHz PRF radar, this equates to a capability of generating approximately 330 targets, all within the radar beam.

For configurations in which the DSP Engine is used to generate clutter, approximately one and one half microseconds are required to complete the calculations for a single range cell. Operating with a one KHz PRF radar, the DSP Engine is capable of generating approximately 665 cells of clutter. For simulations requiring both targets and clutter, a DSP Engine operating with a PRF of 1 KHz, could be configured to generate 100 targets and 465 clutter cells.

Increased target and clutter capacity is available with the use of additional RES DSP and Clutter Engines. As newer and faster DSP components become available, increasingly more dense target and clutter scenarios will be possible.

Summary

The availability of COTS equipment and high performance DSP technology has had a significant impact on the availability of attractively priced, specialized test equipment such as the RES described in this paper. The use of standardized components such as the PC and *LabVIEW for Windows* provides a familiar operating environment, facilitating operator training and also provides a low cost path for future system enhancements.

A Radar Target Generator Architecture Targeted Toward Free-Space Testing of Airborne Radars

Eugene H. Lowe *
Brian J. Donlan

Science Applications International Corporation
429 South Tyndall Parkway, Suite H
Panama City, FL 32404

Abstract

This paper discusses an exciting new radar target generator (RTG) architecture being developed to support free-space testing of modern and future airborne radar systems operating in air-to-air modes. This architecture was developed to fulfill a need at the Avionics Test and Integration Complex at Edwards AFB, CA, and at the Air Combat Environment Test and Evaluation Facility located at the Naval Air Station, Patuxent River, MD. The need is to test highly integrated avionics suites in their installed configuration. The RTG architecture is designed to operate as either a stand-alone RTG or as an element of a large coordinated test configuration. Other elements of the test suite could include CNI, IR, and EW simulators presenting a common multi-spectral test scenario.

Introduction

This paper presents a new radar target generator (RTG) architecture being developed to support free-space testing of modern and future airborne radar systems operating in air-to-air modes. The RTG architecture is designed to operate as either a stand-alone RTG or as an element of a large coordinated test configuration. Other elements of the test suite could include CNI, IR, and EW simulators presenting a common multi-spectral test scenario.

System Architecture Overview

The new RTG architecture provides a stand-alone capability to produce radar target returns to test installed modern radar systems operating in air-to-air modes. It may be installed in an anechoic chamber (e.g., the Benefield Anechoic Facility (BAF) or the Air Combat Environment Test and Evaluation Facility (ACETEF)) or in a ground test laboratory. It is capable of operating via free-space radiation in an anechoic chamber or ground test laboratory without direct connection to the radar system under test (SUT). It is also capable of utilizing SUT data bus information and radar local oscillator samples, and of injecting return signals into the SUT radar.

The free-space RTG architecture has four independent RF channels, each with the capability to provide target skin returns to a SUT radar. In lieu of target skin returns, the channels are capable of generating clutter (main lobe, side lobe, or altitude line) or electronic countermeasures (ECM). The ECM capability includes noise waveforms to test a

radar's susceptibility to selected ECM noise techniques. It also includes range deception, velocity deception, and bin masking techniques.

The free-space RTG consists of the Operator Console/Executive Subsystem, Scenario Models Subsystem, RF Subsystem, SUT RF Interface, SUT Interface, I and GTC Interface, and AOAS Interface. These elements are illustrated in Figure 1.

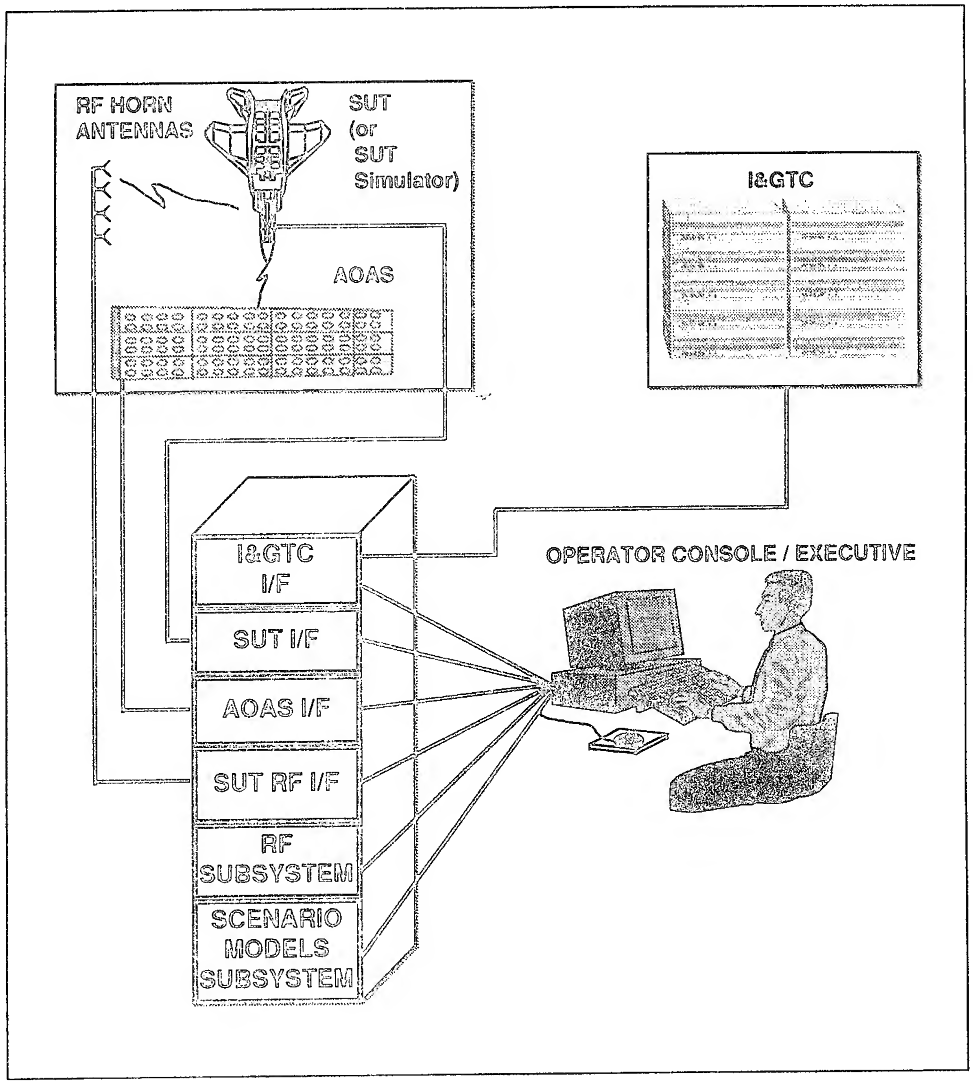


Figure 1 Elements of the free-space RTG

The Operator Console/Executive Subsystem provides the link between the user and the free-space RTG. It performs resource management functions and coordinates the activities of the other subsystems of the free-space RTG system. The Scenario Models Subsystem creates the test scenarios based on the user's inputs. This subsystem contains the high fidelity player models (SUTs, targets, ECM, clutter, etc.), controls the player dynamics, and computes all geometry and hardware interactions in a scenario. It also computes all setup parameters for the RF Subsystem. The RF Subsystem generates the RF radiation in a test chamber in response to the test scenario being executed. The RF Subsystem interfaces with the SUT through the SUT RF Interface by receiving, storing, delaying, modulating, and retransmitting the RF signals through the Horn Antennas (of which four are shown in Figure 1). The proposed

future Angle-of-Arrival Simulator (AOAS) system is also shown in this diagram to emphasize its compatibility with the free-space RTG.

A key element of the new architecture is the use of a replicated shared memory system as the real-time communication medium between the independent processors in the RTG. Each processor in the RTG is a node on the replicated shared memory system. This allows the processing functions to be modularized and grouped optimally within the subsystems for ease of development, testing, and integration. An important consideration in partitioning the system was to have the interfaces transfer only high level parameters, whose formats are then converted by the subsystem using the parameter. Another consideration was to minimize the amount of time critical data passed through the replicated shared memory system, so that data bandwidth and latency problems could be avoided. For instance, local copies of all tables and databases are loaded into the subsystems using them before test execution begins. The architecture and interfaces facilitate adding more processors to run additional models of players. It is also very simple to add independent RF channels to simulate more players, ECM, and clutter.

Architecture Major Advantages

The foundation components and the infrastructure designed for the free-space RTG have the capability of being easily extended to support target generation in other spectra with the addition of spectra-specific hardware and appropriate user interface and real-time software enhancements. As shown in **Error! Reference source not found.**, the workstations that compose the Operator Console/Executive Subsystem and the Scenario Models Subsystem can also be used to control a millimeter wave (MMW) subsystem, an infrared (IR) stimulator, or a CNI stimulator as long as these systems can interface to the replicated shared memory. Through the I&GTC Interface, the free-space RTG is
capable of using real-time commands and data from external simulation environments. The purpose of this capability is to facilitate integrated operation with the future Infrastructure and Generic Test Capability (I&GTC).

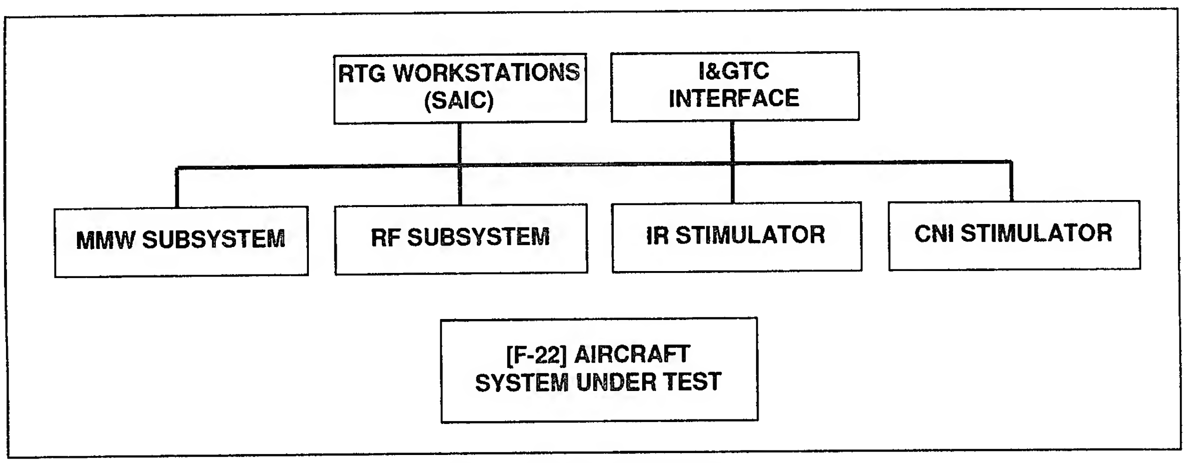


Figure 2 Multi-Spectral Testing

Another major advantage of the free-space RTG architecture is that it permits a high degree of synergy between testing in the BAF, testing in the Integration Facility for Avionics System Testing (IFAST), and flight testing. Figure 3 shows how a common front end (operator console/executive, models, and test scenarios) can be used to conduct tests in one facility and conduct confirming tests in the other, simply by transferring setup files and re-running the test.

By modifying the spread bench F-22 RTG to interface to the replicated shared memory, the free-space RTG RF Subsystem becomes interchangeable with the spread bench F-22 RTG. This permits early testing of the free-space RTG with the F-22 radar by using the F-22 spread bench in the IFAST. It also allows the spread bench F-22 RTG to

be brought into the BAF for installed tests of the F-22 radar. This results in a very powerful and flexible testing capability.

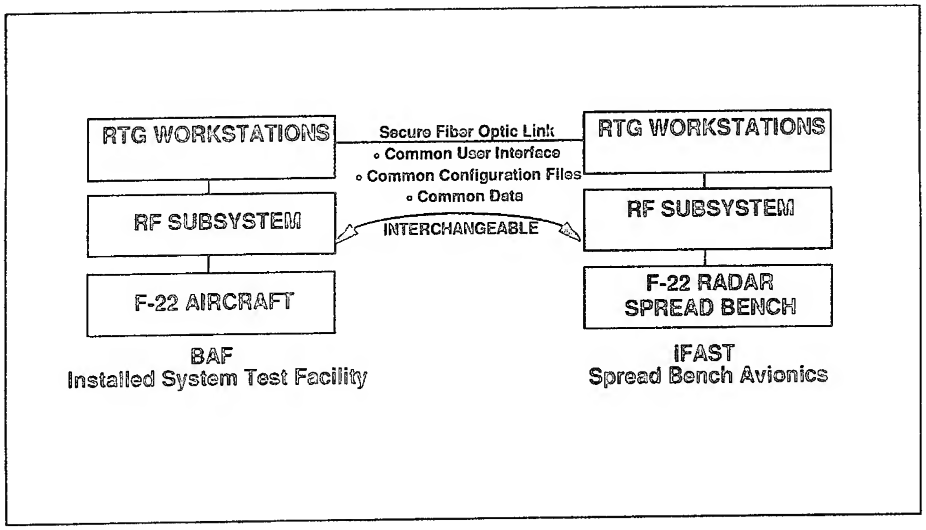


Figure 3 BAF / IFAST / Flight Testing Synergy

An objective of the Installed System Test Facility (ISTF) upgrades is to promote commonality between the ATIC and the ACETEF. Figure 4 shows how the use of common operator console/executive, models, and test scenarios makes this readily possible.

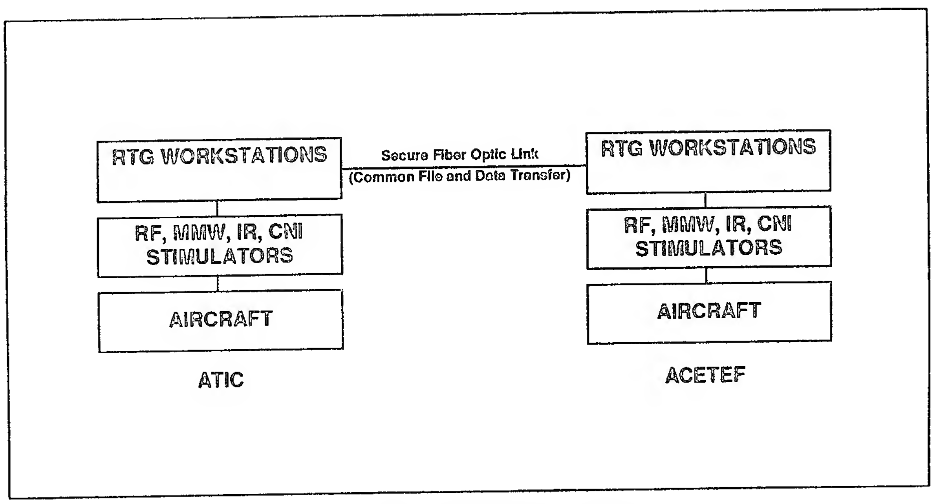


Figure 4 ATIC / ACETEF Testing Synergy

Operator Console/Executive Subsystem

The Operator Console/Executive Subsystem provides a straightforward, intuitive link between the operator (or user) and the free-space RTG. Through Motif-style screen displays and menu options, the user is able to configure, customize, execute, and monitor a myriad of RF test scenarios designed to test and evaluate various radar systems under test. The Executive provides the resource management functions for the free-space RTG. It schedules resources, provides interprocess control, and maintains the compound state of all the free-space RTG subsystems.

Based on a mission scenario, the Executive downloads setup parameters to each of the RF modules and the Scenario Models Subsystem which controls the overall operation of the test. The Executive also formulates commands which coordinate the activities of the RF Subsystem and SUT Simulator during Self-Test and Calibration operations.

The Operator Console/Executive provides intuitive and consistent point and click control over all aspects of a testing cycle; that is, mission development, consistency checking, real-time monitoring of players and hardware, and self-test and calibration of subsystems. The Operator Console/Executive allows the working environment (or tool) to be selected by a single mouse click. All working environments provide on-line context-sensitive help. The Operator Console/Executive allows the user to quickly build scenarios, calibrate the free-space RTG hardware, and set up tests and simulated missions. The visual technology provided with this system lets the user "drag and drop" (that is, select an icon and drop it onto an appropriate form drop point) a wide assortment of objects like targets, jet engine modulation (JEM), clutter, calibration tables, radar cross section (RCS), etc.

A scenario can quickly be built using the drag and drop feature. Mission flight path, JEM, and target characteristics objects are acquired from the library function, edited as necessary, and assembled to quickly create a test scenario. This easy-to-use scenario definition process allows multi-target scenarios to be built in minutes. With the click of a button, the user can display a powerful satellite view of a scenario in progress. The satellite view may also be used to preview a scenario without having to receive or generate RF. By clicking on a player object that is displayed in a scenario satellite view, the user can observe or change the real-time parameters of that player; another click and the selected parameters are displayed in a strip chart. To set up a test, the user simply employs the mouse to select from the library a preconstructed and previewed scenario which contains all of the elements to set up a mission. Any setup function is just a mouse click away. For example, clicking on the download button downloads the scenario setup. The user is prompted when the download is complete and may then click on the start button or any other control button to further control the operation of the free-space RTG.

Scenario Models Subsystem

The Scenario Models Subsystem (SMS) simulates and controls the free-space RTG test scenario. It contains a high fidelity computer model of each airborne object (e.g., aircraft and missiles) in the test scenario and computes their position, flight path, and orientation throughout the scenario. At the specified update rate, the relative position, aspect angle, return signal amplitude, range, and range rate of each of these objects, with respect to the SUT, is calculated. These values are formatted and provided at this update rate to the RF Subsystem, which uses them to generate a synthetic RF environment for the SUT radar.

In addition to target returns, the SMS computes the dynamic clutter and ECM parameters. The Scenario Models Subsystem uses the altitude of the SUT, antenna pattern, and beam pointing angle to compute the location and size of the footprint on the ground of the main beam of the SUT radar. This, together with SUT velocity, is used to compute the effects of clutter (main lobe, side lobe, and altitude line) on the return signal to the radar. The SMS also calculates the dynamic parameters for the specific ECM techniques currently selected. The computed clutter simulation parameters and the dynamic ECM parameters are sent to the RF Subsystem, which uses them to generate these elements of the synthetic RF environment.

Another function of the Scenario Models Subsystem is to provide computed SUT flight path information to the SUT Inertial Navigation System (INS) (via the SUT Interface) to update the position and orientation of the SUT in the scenario.

The modular, object-oriented design of the Scenario Models Subsystem stresses high fidelity models and flexibility in the choice of models to use in any particular test setup. All of the scenario models run as independent processes communicating via the replicated shared memory system. This allows SUT and target models, as well as models being executed in an external scenario, to be incorporated without modification to the other elements of the free-space RTG. Multiple SUTs are supported, and additional processors can be added modularly to provide increased computational power and speed as needed.

A feature that is significant to the fidelity of the SUT and target models is that the Scenario Models Subsystem is based on the Joint Modeling and Simulation System (J-MASS) architecture, optimized for real-time performance. The J-MASS architecture is in turn based in part on the Weapon System Interactions Model (WSIM). The WSIM architecture, developed for the DoD, successfully demonstrated independent player update rates with synchronous geometry updates. This is very important for models running in the I&GTC for use by the free-space RTG. J-MASS extended the WSIM architecture by allowing player instances, or groups of player instances, to be independently compiled and linked into separate programs. These programs execute concurrently, utilizing multiple processors if they are available. This allows multiple processors to be utilized efficiently in a language-independent implementation of J-MASS. Individual player models may be in any computer language as long as they conform to the interconnect protocol.

RF Subsystem

The RF Subsystem function within the free-space RTG system receives the SUT transmissions via free space, determines the radar transmission characteristics, and faithfully reproduces target and clutter returns which simulate air-to-air (A/A) engagements between an interceptor aircraft and one or more penetrator aircraft. To provide these near real world simulated target and clutter signals, the RF Subsystem architecture is designed to respond to commands provided by the Operator Console/Executive Subsystem and Scenario Models Subsystem. The RF Subsystem converts these commands, which are based on the SUT and player dynamics, into appropriate time delays and modulations to depict radar cross section, range delay, range attenuation, Doppler offset, and jet engine modulation of target signatures. In addition, the RF Subsystem computes the appropriate modifiers to simulate scintillation, clutter, and ECM jamming signals.

Figure 5 shows a block diagram of the RTG system which emphasizes the RF Subsystem. The RTG RF Subsystem consists of a SUT interface, down converter, fast agile synthesizer, generators, and up converters. Each of these components is described below. The RTG also contains a SUT simulator, which is used for test and calibration of the RTG.

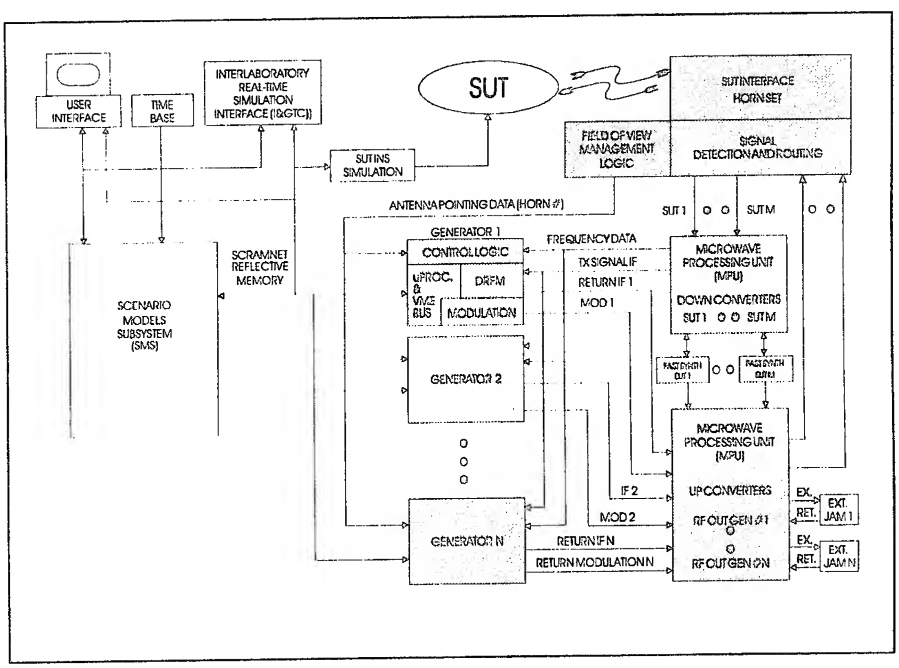


Figure 5 RTG System Block Diagram

The RTG is also capable of operating in a partially or fully connected configuration. This may be useful in situations where a higher level of performance is achievable using direct connection to the SUT and the connections of interest are easily accessible under the allowed simulation conditions. In installed system testing, it is not always possible to gain access to signals from the radar under test. However, when these signals are available, they may allow the simulator to achieve a higher level of performance due to the information they contain about the radar operating conditions.

It may also be useful to operate all or a portion of the RTG in a direct injection mode during multiple SUT testing to facilitate isolation of signals between the SUTs. For example, a coordinated test between an aircraft and a missile could be performed where the transmission and reception paths are direct connected for one of the two SUTs and free-space for the other. This eliminates any potential problems with the simulator sorting the signals from the two radars (assuming they are frequency coincident). Similarly, it eliminates the possibility of one SUT seeing the returns intended for the other SUT. This hybrid configuration allows the interaction of the two assets to be tested without complications due to unwanted interference.

The SUT RF Interface detects the SUT free-space transmission and determines which antenna is being illuminated. It routes the received RF signal to the RTG Down Converter and routes the RTG Generator/Up Converter channel output to the horn antennas for retransmission to the SUT.

The Down Converter receives the RF SUT transmission signal from the SUT RF Interface and translates that RF signal to base band. The Down Converter measures the SUT transmission frequency and provides this information for use within the RF Subsystem. It provides RF signal amplitude dynamic range compression using an Automatic Gain Control (AGC), makes the controls available for setting the fast agile synthesizer, and furnishes several local oscillator signals used in the up and down converters. One Down Converter provides the input signal for all generators.

Each of the Generator assemblies consists of a digital RF memory (DRFM) kernel, generator processor, reflective memory, global memory, Doppler calculator, data handler, Doppler module, arbitrary waveform generator, noise filter modules, and attenuator driver/parallel I/O card. These waveform processors and modules are used to provide the control signals to develop the simulated target and clutter signatures. The specific parameters such as range delay, Doppler offset, amplitude, scintillation, JEM modulation, clutter, and ECM jamming characteristics are implemented as commanded by the Scenario Models Subsystem.

The Up Converter adds the appropriate modulations to the delayed replica of the SUT RF signal using commands provided by the generator signature processing modules. The Up Converter generates main lobe, side lobe, and altitude line clutter using control signals from the generator processor modules. This component up-converts the intermediate frequency (IF) target and clutter signals to the SUT transmission carrier frequency. It provides the RF output interface for the following: free-space transmission back to the SUT via the SUT RF Interface assembly (i.e., programmable switcher/combiner and horn antennas), direct injection into the radar RF front end, and external ECM asset interface.

To facilitate active two-way free-space testing in the BAF, the free-space RTG design uses four dual-polarized, quad-ridged horns which can be positioned manually in any desired location within the anechoic chamber. Polarization, either vertical or horizontal, is user selectable. A switcher/combiner unit is used and is housed in a separate assembly located outside the chamber area close to or within the RTG rack assembly. The switcher/combiner performs: fast detection of SUT antenna beam illumination, routing of the received RF signals to the RTG, and routing of RTG output signals to the horn antennas for retransmission to the SUT. The single horn antennas are shared for RTG transmission and reception of SUT transmissions.

SUT Interface

The SUT Interface contains the signal and data interfaces and formatters that link the free-space RTG with the SUT. The Scenario Models Subsystem provides SUT flight path information to the SUT Inertial Navigation System (INS) via this interface. This INS information updates the position and orientation of the SUT in the scenario, and is

necessary in all operating configurations. When connections to the radar data buses are allowed, this interface will receive, format, and provide to the free-space RTG real-time information on the radar mode, transmit frequency, modulation waveform, and beam position. When RF connections to the SUT radar are allowed, this interface will receive buffered samples of the radar local oscillators and timing signals for use by the free-space RTG.

SUT Simulator

The SUT Simulator has two purposes: (1) generate RF signals to support engineering tests and calibration of the free-space RTG, and (2) monitor the free-space RTG outputs for engineering purposes. The monitoring function can also be used during the SUT testing to verify correct operation of the free-space RTG. The SUT Simulator will consist of rack-mounted commercial RF test equipment in a control room and a high power amplifier and receive/transmit antenna located in the anechoic chamber. The SUT simulator can be set up and controlled either manually or via a GPIB bus from the Operator Console/Executive Subsystem.

I&GTC Interface

The I & GTC Interface provides the future capability for accepting control commands and real-time data from external simulation environments.

AOAS Interface

The AOAS Interface provides the interface between the free-space RTG and a proposed Angle-Of-Arrival Simulator (AOAS) system. The AOAS system will provide dynamically moving targets in azimuth and elevation via a large wall of steering antenna arrays.

System States

The free-space RTG has four distinct operating states. These are:

- **Standby**
- **Self-Test and Calibration**
- **Stand-Alone**
- **Integrated**

An operating mode is entered by selecting its icon on the main menu of the user screen. In the **Standby State**, the user can enter and work in the editing environment. The user may also download previously generated calibration files, and can configure and verify the status of the free-space RTG hardware. In the **Self-Test and Calibration State**, which is accessible by clicking on either the BIT or the CAL icons, the user may run system calibration and specify the files in which the calibration tables are to be stored. The user may also perform built-in test (BIT) operations and display the results in a Motif window. If a failure is detected during a BIT, a report is issued to the user indicating the nature and location of the failure, and, whenever possible, identifying the suspected component. When the free-space RTG is in the **Stand-Alone State**, the user may set up a SUT radar test by defining the target characteristics (e.g., RCS, JEM, scintillation, etc.) and mission flight profiles for each target. The user can also set up and run a non-scenario test of the SUT radar. This operating mode also allows the user to recall, modify, and run any previously stored test setup. The **Integrated State** is provided for integrated operation with the I&GTC.

Summary and Conclusions

The new target generator architecture provides a free-space simulation environment for the configuration testing of installed radar systems. The free-space interface to the SUT allows for non-intrusive testing and rapid transitions between different radar system types. Incoming signals from the SUT are analyzed to determine the waveform

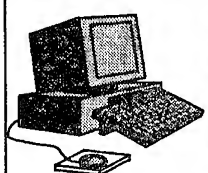
characteristics and the antenna pointing angle. The signals are then stored in a group of return generation channels. After the appropriate time delay, the signals are reconstructed and modulations are applied to simulate target, clutter, and ECM returns.

The free-space RTG is also capable of operating in a partially or fully connected configuration. This may be useful in situations where a higher level of performance is achievable using direct connection to the SUT and the connections of interest are easily accessible under the allowed simulation conditions. In installed system testing, it is not always possible to gain access to signals from the radar under test. However, when these signals are available, they may allow the simulator to achieve a higher level of performance due to the information they contain about the radar operating conditions.

It may also be useful to operate all or a portion of the RTG in a direct injection mode, especially during multiple SUT testing to facilitate isolation of signals between the SUTs. For example, a coordinated test between an aircraft and a missile could be performed where the transmission and reception paths are directly connected for one SUT and are free-spaced for the other. This eliminates potential simulator problems in sorting the signals from the two radars (assuming they are frequency coincident). Similarly, it eliminates the possibility of one SUT receiving returns intended for the other SUT. This hybrid configuration allows the interaction of the two SUTs to be tested without complications due to unwanted interference.

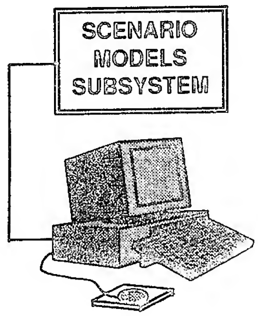
The following figures summarize the features and capabilities of the free-space RTG system.

OPERATOR CONSOLE / EXECUTIVE



- **RTG Control**
- **Test Setup**
- **Test Monitoring**
- **Real-Time Instrumentation Data Collection**
- **Scenario Building**
- **Execute Self-Test**
- **Execute Automatic Calibration**
- **Control & Monitor Diagnostic / Troubleshooting Tests**
- **Test Data Analysis**
- **Point & Click User Interface**

Figure 6 Operator Console/Executive Subsystem Features



- Calculates
  - SUT Flight Model
  - Target Flight Model
  - Environmental Model
  - SUT / Target Geometry
  - Clutter Model
- Computes Setup & Dynamic Update Parameters for RF Channels
  - Target Return
  - ECM
  - Clutter
- Open Architecture Supports:
  - Multiple Types of Processors
  - Multiple Languages
  - Multiple Operating Systems
  - Many Existing Models

Figure 7 Scenario Models Subsystem Features

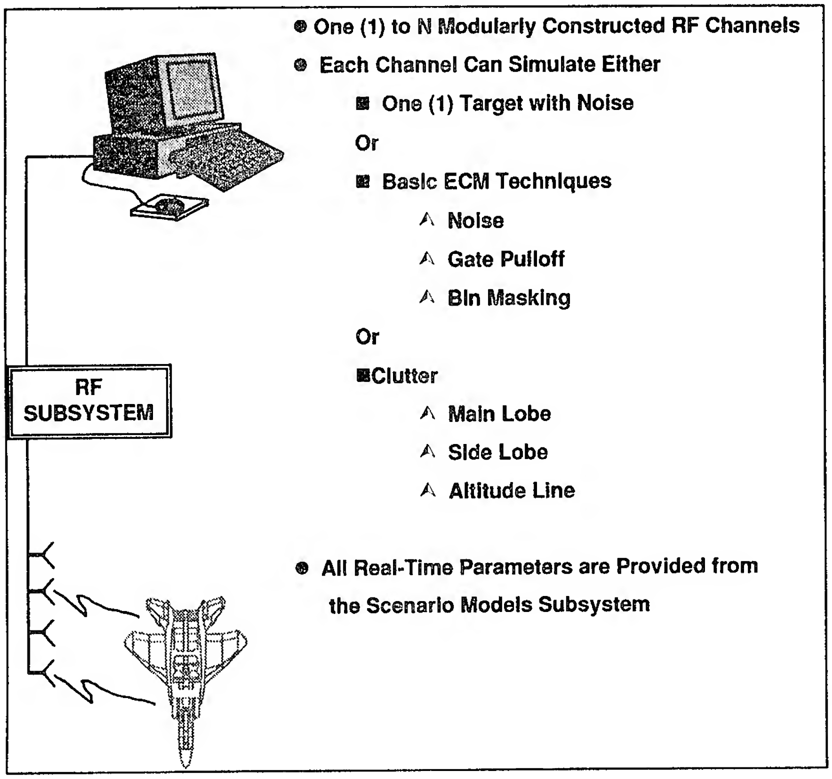


Figure 8 RF Subsystem Features

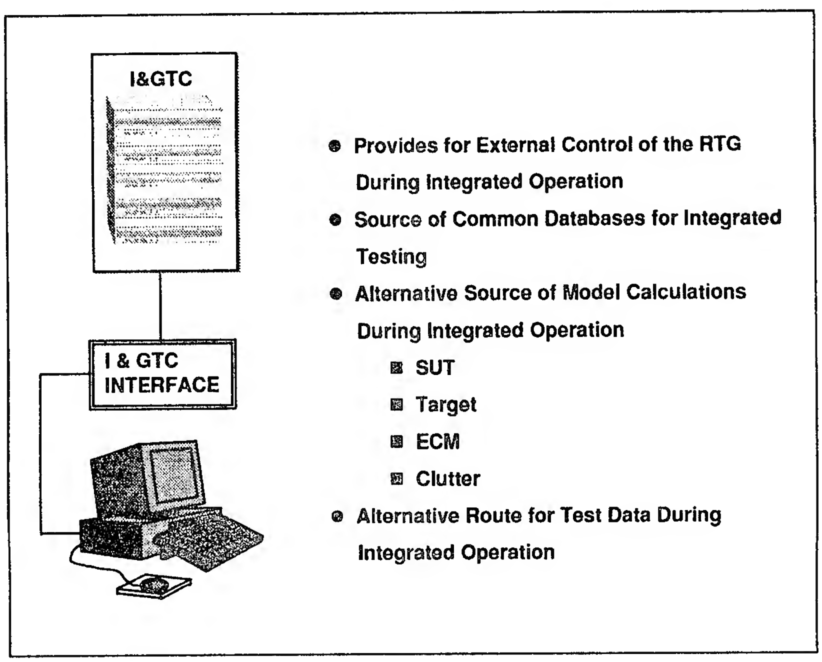


Figure 9 I&GTC Interface Features

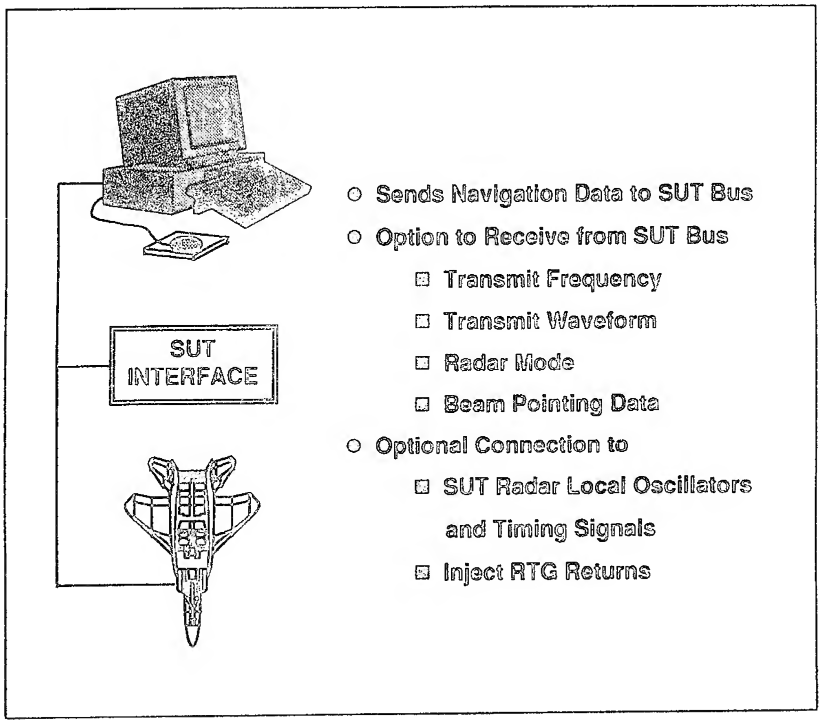


Figure 10 I&GTC Interface Features

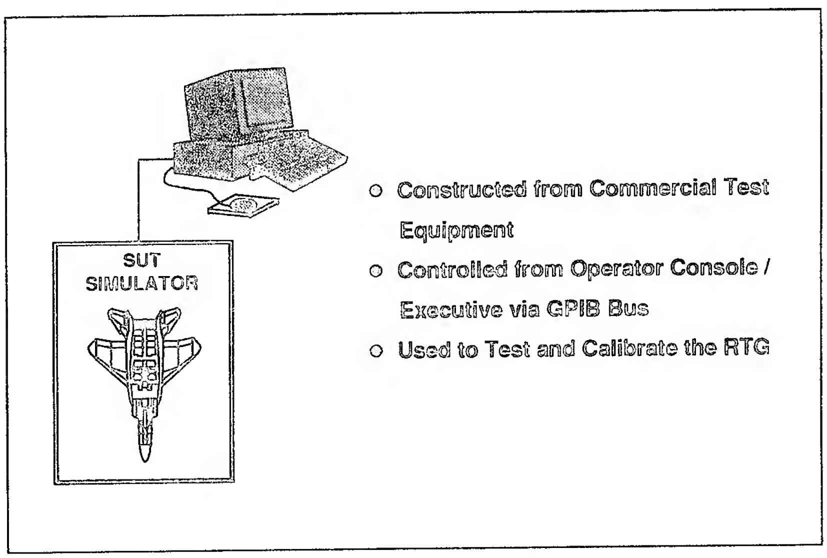


Figure 11 SUT Interface Features

ECM Waveform Generator

Gary Luper
Engineer Specialist Senior
Northrop Grumman Corporation
Electronics and Systems Integration Division
600 Hicks Road
Rolling Meadows, IL 60008
Phone: (708) 259-9600 ext 4559
Fax: (708) 870-5748

Abstract

ECM techniques are usually established by analytical methods based on threat radar parameters. However, to verify and maximize the effectiveness requires many technique parameter variations and subsequent RF test iterations against actual and simulated threat radar hardware. Many such test environments require a technique generator that can be set up quickly, provide a wide selection of techniques, and allow for simple modification of technique parameters on-site. A field-portable and programmable ECM Waveform Generator (EWG) has been developed by the Northrop Grumman Electronics and Systems Integration Division to aid in ECM technique development and evaluation. The EWG operates between 2 GHz and 18 GHz with a selectable 20 or 100 MHz response bandwidth. Digital RF Memory (DRFM) technology is used to generate coherent techniques such as multiple false targets, RGPO, VPGO, coordinated range/velocity pulloffs, as well as broadband noise techniques. The EWG system is self-contained in a VME chassis occupying approximately 4 ft^3. Techniques are easily generated using a graphical user interface (GUI) based software and a laptop PC via a RS-232 interface. The user can select various ECM techniques and quickly modify any parameter of a given technique during test to optimize that technique's effectiveness. A breadboard of this system has been used in multiple tests against threat radar systems with successful results. In addition, the EWG provides a low cost waveform generation capability that can supplement characterization of foreign military exploitation (FME) hardware. This paper contains material previously released and in the public domain.

Ground Mount Application

A ground mount test is an efficient and systematic method of verifying and optimizing an analytical ECM technique against a threat radar or simulator, prior to flight test qualification. The test radars are instrumented to evaluate the ECM technique effects on particular functions of the radar (e.g., range or angle tracking circuits, AGCs, etc.). The effectiveness of any technique is determined based on the collected data, as well as its influence on radar operations when applicable. Multiple test iterations are performed with some parameter adjustments to optimize a potential technique. The test results often provide insight for alternate techniques.

The radars and simulators are typically installed at fixed locations and are not easily moved since there is a substantial effort required to setup and support. An ECM test requires the ECM equipment to be transported, setup, and operated at the ground mount site. The test usually involves using an operational ECM system that is designed for an aircraft installation which requires auxiliary equipment to make it functional at ground mount site (400 Hz AC power, support computers, etc.). There is a substantial cost involved to configure and program the ECM system with the technique waveforms prior to and during the test. It is sometimes difficult to make significant changes to ECM techniques at the site that have not been verified in the laboratory. The inflexibility of the system may produce incomplete test results.

The EWG system was proposed as laboratory/field test equipment that can easily reproduce ECM techniques from analytical waveforms. The system has a direct application to ground mount tests because the system can be set up quickly, generate techniques defined by the user, and allow for modifications of technique parameters on-site. In addition, the reduction in funding to conduct ECM developmental testing has driven the need to reduce the test cost and more efficiently utilize test time. The EWG suitably addresses this objective since it does not have the overhead cost typical of conventional ECM systems. A typical ground mount configuration using the EWG is shown in Figure 1.

EWG Description

The EWG is a single-channel ECM system that operates between 2 and 18 GHz with selectable 20 or 100 MHz response bandwidths. The equipment has standard repeater modes and non-coherent transponder modes. In addition, a signal memory unit (SMU) is utilized to generate techniques such as multiple false targets, RGPOs, VPGOs, and coordinated range/velocity pulloffs. The EWG does not have a receiver or tracker, therefore synchronization to the PRI or scan requires TTL trigger signals from the radar under test. This is not considered a severe limitation for technique evaluation since the radar parameters are known and programmed prior to test. An overview of the EWG parameters is shown on Table 1.

The EWG hardware is self-contained in a VME chassis consisting of three RF modules and two digital modules. It occupies approximately 4 ft^3 and is powered by a standard 120 VAC source. The EWG VME format allows for future expansion, upgrades, or field interchangeable modules for specific hardware configurations (i.e., another RF channel, a VME target generator, or a VME synthesizer). Additional equipment required to support the EWG is a Laptop PC and a HP Spectrum Analyzer with a IEEE-488 interface. The Spectrum Analyzer is used for several activities during a test: it serves as a receiver to tune the YIG LOs in the RF tuner module, provides a direct approach to measure J/S, and allows user verification of the selected ECM waveform. The EWG architecture is processor-controlled to allow the flexibility to generate and modify a variety of techniques.

A block diagram of the EWG is shown in Figure 2. There are two parallel RF paths that form a typical repeater system; one RF path is for the receive section and the other is for the transmit section. For repeater and coherent DRFM techniques, the coherency of the signal is maintained by utilizing the same LOs for each up/down conversion stage. It should be understood that a coherent response in this paper means the signal is coherent in frequency only and not by the strict definition of coherent in frequency and phase. Below is a brief description of each module of the EWG system.

Table 1: EWG Parameters

Parameter	Value		
Frequency (GHz)	2 to 18		
Response Bandwidth (MHz)	20 or 100		
Receive Signal Power (dBm)	0 to -45		
Transmit Power (dBm)	0*		
Response Delay (nsec)	Repeater	Transponder	DRFM
	140	100	230/900**
Gain (dB)	45	---	45

* Prior to external HP Amplifier with approximately 22 dB gain.
** Delay is dependent on DRFM technique. The 900 nsec delay is for velocity and phase modulation techniques only.

The primary function of the Block Converter is to extend the operating bandwidth of the EWG to operate between 2 to 18 GHz. The Block Converter converts the receive/transmit signals between 2 to 6 GHz to/from the RF tuner band of 6 to 18 GHz. The signals operating between 6 to 18 GHz are essentially passed through the Block Converter directly. The RF path is software selected based on the programmed radar frequency.

The RF tuner VME module consist of two subassemblies, the RF tuner and the IF module. The RF tuner subassembly contains a tunable YIG LO to down convert the receive signal to the first IF and up convert the first IF to the Block Converter RF band. The LO is a digitally controlled YIG oscillator which has an extremely stable frequency attained by a closed loop tuning method. The IF module performs the second IF conversion and selects either a 20 or 100 MHz filter bandwidth. RF switches are configured for the selected response, which include coherent repeater, non-coherent transponder, AWGN noise, or signal memory unit, SMU, modes. The implementation of AM waveforms is also performed in the IF module.

The SMU module contains a DRFM that can record and replay detected signals at IF and an I/Q modulator that supports velocity and phase modulation techniques. The DRFM digitizes the IF signal using one of two sampling frequencies and stores the 'recorded' signal in memory. The selections are single bit sampling at 540 MHz or ten bit sampling at 60 MHz

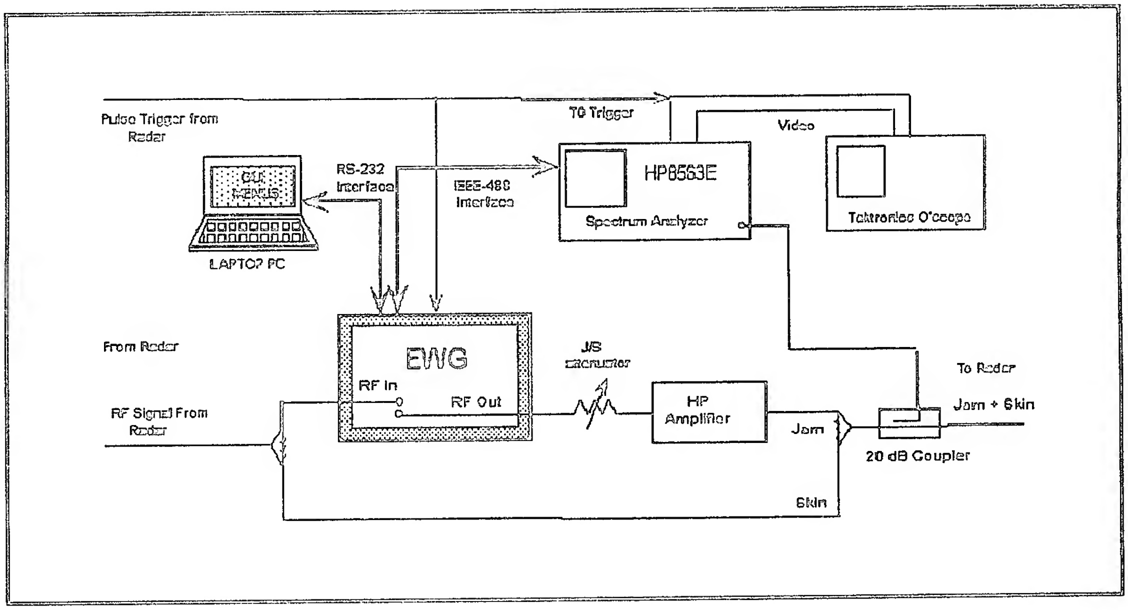


Figure 1: Typical EWG Ground Mount Set Up

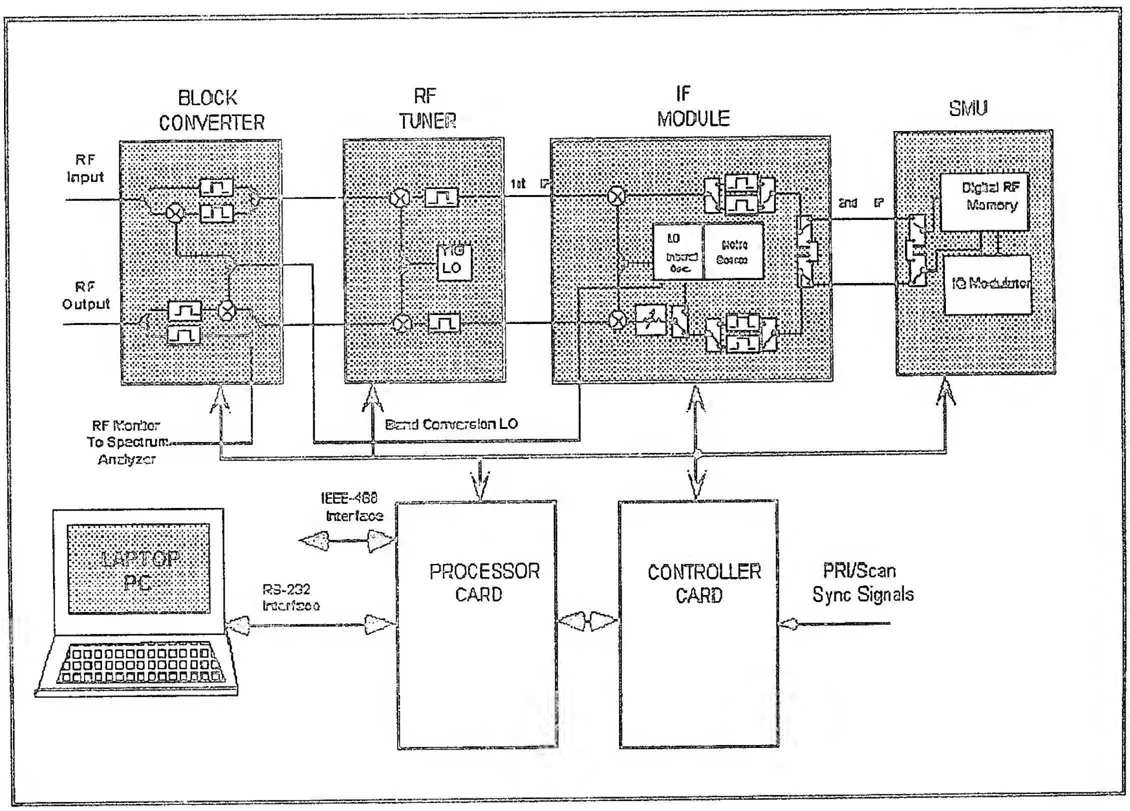


Figure 2: EWG Block Diagram

for wide or narrow band applications, respectively. The DRFM has programmable modes that provide signal processing and timing control applications to create a variety of techniques.

The two digital modules are the Processor Board and the Controller Board. The Processor Board contains a dedicated MC68040 processor with embedded operating software. The Processor Board could operate the EWG as a stand-alone system except that the Laptop PC is required to communicate the technique requirements over a RS-232 interface. The Controller Board is a MC68332 programmable system module that manages the timing and control signals to the RF modules for the selected user-defined ECM technique. The Controller Board contains four independent timing and control channels.

The primary function of the laptop PC is to create or modify the ECM technique and download to the parameters to a EWG processor board. The techniques are generated using GUI menus on the laptop PC. The menus for the technique parameters are 'standardized' such that any user familiar with ECM techniques can generate a waveform. The user can create and modify any waveform, and save/retrieve the technique on disc. The laptop can be used either as an interactive interface, or as a system controller. In the interactive mode, the processor card configures the system for the selected technique and provides the system status to the laptop. The interactive mode permits the laptop to be available for other applications. In the system controller mode, the laptop controls the EWG system for diagnostic routines or manually controlled ECM responses. All technique waveforms reside either on the laptop hard disc or a floppy. The laptop can be disconnected from the EWG at the end of a day's test and made available for other tasks (e.g., analyzing and writing up test results). The techniques can be secured on disc at the end of the day without securing the equipment.

Where possible, the software has been structured to generate selected techniques from 'standardized' variables using function-oriented subroutines. Figure 3 is an example of a coordinated Velocity/Range Pull Off generated from user defined variables. This example is a mathematical representation of the functions required to generate the response sequence. The sequence of functions define at any point in time the range response delays or doppler frequency shift. The timing strobes and the IQ doppler words are computed from these functions relative to the PRI and stored within the Controller Board.

Technique Capabilities

Conventional ECM techniques, such as repeater and noise techniques, can be generated without the use of the SMU hardware. However, the SMU/DRFM significantly expands the capabilities of the EWG. A digital representation of the IF signal is stored within the DRFM and can be replayed at any time afterwards. Timing is controlled either by a selected mode of the DRFM or directly with the Controller Board. The digitized signal can also be processed through an IQ modulator to produce frequency or phase modulated responses. Table 3 summarizes the EWG technique capabilities. Specific details of the SMU mode techniques are shown in Table 4.

Figure 3: Example of Function Generated Pull Off Timing Sequence from User Defined Variables

Define Pull Off Parameters: The GUI menus require the technique variables

Pulloff_type 1	Pulloff type: Parabolic Pulloff=1, Linear Pulloff=0
Pulloff_direction 0	Pulloff direction: Inbound pull=1, Outbound pull=0
Init_pulloff_dwell_time 1·sec	Initial pulloff dwell time
End_pulloff_dwell_time 2·sec	Ending pulloff dwell time
Pulloff_duration 5·sec	Linear Pulloff duration (computed for parabolic pulloffs)
Pulloff_distance 10·usec	Pulloff distance (as time delay) of ECM response with respect to target echo
Max_pull_rate 2·g	Number of g's (for parabolic pulloff only)

Functions for Linear and Parabolic Pull Offs: (An inbound pull is dependent upon a fix PRI waveform)

$$\text{Velocity\_RGPO}(t)\quad \text{if}\left(\text{Pulloff\_type}=1,\text{Max\_pull\_rate}\cdot t, c\cdot\frac{\text{Pulloff\_distance}}{\text{Pulloff\_duration}}\right)$$

where; c is speed of light

$$\text{Pulloff\_duration}\quad \text{if}\left[\text{Pulloff\_type}=1,\sqrt{\frac{c\cdot\text{Pulloff\_distance}}{\text{Max\_pull\_rate}}},\text{Pulloff\_duration}\right]$$

$$\text{t\_delay}(t)\quad \frac{\text{Velocity\_RGPO}(t)}{c}\cdot t$$

$$\text{Doppler\_shift}(t)\quad \frac{-2\cdot(\text{Velocity\_RGPO}(t))\cdot\text{freq\_rcv}}{c}$$

Timing Sequence of waveform as specified

t1 Init_pulloff_dwell_time + Pulloff_duration

t2 t1 + End_pulloff_dwell_time

time_in_sweep(t) mod (t, t2)

T_dly3(t) if (t < t2, Pulloff_distance, 0·sec)

T_dly2(t) if (t < t1, t_delay(t − Init_pulloff_dwell_time), T_dly3(t))

T_dly1(t) if(time_in_sweep(t) < Init_pulloff_dwell_time, t_delay(0·sec), T_dly2(time_in_sweep(t)))

T_dly(t) if(Pulloff_direction = 0, T_dly1(t), pri − T_dly1(t))

Dop1(t) if (t < t1, Doppler_shift(t − Init_pulloff_dwell_time), Doppler_shift(Pulloff_duration))

Doppler(t) if (time_in_sweep(t) < Init_pulloff_dwell_time, 0 Hz, Dop1(time_in_sweep(t)))

Resultant Technique Timing for Programmed Variables

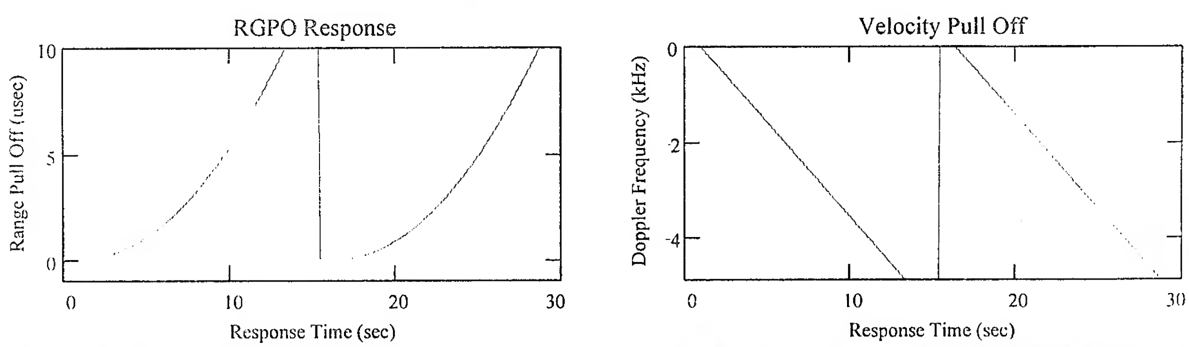


TABLE 3: EWG Techniques		
COHERENT TECHNIQUES		
Repeater		
SMU:	MFT	Fixed
		Pull Offs, independently programmed
	Head to Tail Playback	with phase adapt
		without phase adapt
	Pull Offs	RGPO
		VGPO
		coordinated range/velocity
	Range Cover Pulse	Centered on Skin Pulse
	Velocity Techniques	Cover Pulse
		Velocity Noise
		Velocity False Targets
NON-COHERENT TECHNIQUES		
Internal Oscillator	Range Cover Pulse	
Noise	RGN Cover Pulse	20 MHz or 100 MHz
	Barrage/Spot	
	SMU modes referenced above	MFT
		Pull Offs
INDEPENDENT WAVEFORMS APPLICABLE TO ABOVE TECHNIQUES		
AM	Linear and/or ASWM	freq range= 1 to 10 kHz
		sweep duration= 1 to 30 sec
		max rate= 100 Hz/sec
		triangular or sawtooth
		Linear max DOM of 25 dB
		ASWM max DOM of 40 dB
	Blinking	
Cover Pulse Options	Pulse Width	0.1 to 40 μsec, 0.1 μsec step size
	Fixed Position	Leading Edge
		Centered
		Programmable
	Wobbulated	Position or Pulse Width
		Triangular or Sawtooth Sweep
		1 to 20 sec sweep, 1 sec step size
		0-100% Pulse Width,1% step size

TABLE 4: Summary of SMU technique functions and specifications		
Cover Pulse	Addressed	< radar pulse width
	MFT	> radar pulse width
	Head to Tail Playback	
RPGO	Linear or Parabolic	
	Inbound or Outbound	
	Duration	1 to 30 sec
	Pull Rate	1 to 20 G's
	Max PO distance	50 μsec
	init. or end dwell	0 to 10 sec
VGPO	Linear, Parabolic, or Cubic	
	Pull rate	1 to 200 kHz
	Pull Range	±300 kHz
	Pull through carrier	
CRVPO	reference RGPO and VGPO specifications	
MFT	Fixed Position or Pull Off	
	Max No of False Targets	256
	Independent RGPOs	
VFT	No. of Targets	4
	Position	±300 kHz
VNS	2048 individual tones	
Polyphase	4096 phase shift values	
	0 to 360°, 5° resolution	

The EWG can be used for atypical techniques based on control signals from the radar under test. For example, an inbound RGPO is an impractical technique to counter pseudo-random stagger pulse radars. However, this type of technique may be useful for the evaluation of the radar operation. The EWG can be configured to use a pretrigger from the radar, such as the main bang at T_0, for the generation of this technique.

Field Test Results

A prototype EWG system was demonstrated in three ground mount tests earlier this year. The prototype contained the RF hardware, but the technique processing and control were

performed using a HP computer and a two pulse generators. Two of the tests were performed against SAM radars and one was against an AI radar. One test was conducted over a two week period accumulating data for approximately 350 runs using 43 techniques. Due to the timing and control limitations of the equipment, some of the techniques suggested during the test were not possible.

The EWG was powered up and calibrated within fifteen minutes. Creation of the techniques on-site were typically programmed in the morning to support the day's test. This occurred during the radar warm up period. The time required to change techniques between runs was typically about two minutes. This time duration was dependent on the specifics of the techniques, operator interface, and the hardware limitations of the prototype. The current VME version is capable of technique changes in less than 15 seconds and the GUI menus and technique files should expedite the user interface. Also, the same timing and control limitations do not exist on the VME version, as with the prototype, with the incorporation of the Processor Board and the Controller Board.

Summary

The EWG offers an alternative for evaluating ECM techniques in ground mount test and features the following benefits: less support equipment, portability, easy setup, and on-site techniques programmability from standardized menus. The EWG system is a processor structured architecture to provide flexibility for technique generation. The test engineer can generate a variety of techniques that are representative of an array of ECM systems, allowing any particular ECM contractor or agency to verify a potential technique. In addition, a significant advantage of the EWG is being able to generate techniques not supported by conventional ECM systems, to aid in the development of potential system upgrades.

The VME format allows for upgrades to the EWG that can further its technique applications (i.e., cross pol, cross-eye, simultaneous dual beams).

A prototype of the EWG has been successfully used in the field. Although the system did not have the full complement of timing features, it demonstrated the practical benefits of using such a system during ground mount testing. Overall, the number of techniques and variants that were tested demonstrated that the EWG provides a valuable capacity for characterizing FME systems, and developing and evaluating techniques against threat radars.

Common User Interface for Radar Target Generators

*Tom Haugh, Dick Nelson**, and *Sherry Reighter*

Science Applications International Corporation
Test Support Division
429 S. Tyndall Parkway, Suite H
Panama City, Florida 32404
Richard.Nelson@cpmx.saic.com

Abstract

The Director system is a multi-spectrum target generator test engineering environment developed by SAIC. Director is a common user interface and operating environment for target generators. It is composed of toolkits for real-time simulation; research and development; mission planning, control, and monitoring; scenario rehearsal; and real-time data logging. Director optimizes engineering resources by providing a consistent interface for scenario development and for scenario, hardware, and instrumentation setup, control, and monitoring over multiple target generators.

A common test engineering environment reduces the training time required to become proficient with working environments, and Director's intuitive drag and drop editor significantly reduces the time needed to build complex test scenarios. Director also offers the additional economical advantage of reusability. Many components of a test scenario can be made interchangeable over a variety of target generators; for example, flight paths, ECM setups, etc. Even complete scenarios can be reused. Testing can also be decreased by reusing proven components. Director provides test configuration control for each test scenario developed. This allows entire mission setups to be evaluated or rerun with a high degree of confidence.

Executive Summary

A target generator is often required to stimulate a modern radar system when testing is conducted in an anechoic chamber or a ground test laboratory. An efficient, user-friendly simulation environment allows the tester to easily interact with the target generator to control and monitor the execution of a test scenario. A straightforward, intuitive user interface (or test engineering environment) enables the tester to quickly configure, customize, execute, and monitor the test scenarios. Significant advantages accrue if test engineers use a common user interface which is spectrally independent and capable of controlling a wide variety of generators (RF, MMW, IR,

etc.). This allows easy transfer of tests from one test facility to another without recreating the entire test setup.

Director is an engineering environment developed for radar target generators that is equally applicable to generators in other spectra. The architecture of this common test engineering environment employs cutting edge technology; e.g., Motif Drag-and-Drop paradigms on DEC Alpha workstations, Ada, SAIC's Director programming environment, and Systran Corporation's Shared Common Random Access Memory Network (SCRAMNet). This combination of unique architecture and state-of-the-art technology makes this engineering environment for radar target generators a system that can easily be expanded as technologies grow, can be readily integrated for use with various back ends (including current target generators, with appropriate modifications), and can be easily used with other spectra to lower the costs of target generator development.

Director has a common parameter database that the tester can manipulate, feed to a hardware-specific compiler for validation, and then send to a specific radar target generator. Through Motif-style screen displays and menu options, the user is given intuitive and consistent point-and-click control over all aspects of a testing cycle; e.g., mission development, consistency checking, real-time monitoring of scenario players and hardware, and calibration and self-test of hardware subsystems. The visual technology provided with this system lets the tester "drag and drop" a wide assortment of objects (e.g., targets, clutter, calibration tables, etc.) to configure a target generator for a specific test. This feature allows multi-target scenarios to be defined very quickly.

Common Test Engineering Environment

More than just an operator console, the Director architecture includes an Executive that provides the resource management functions for the target generator. It schedules resources, provides interprocess control, and maintains the compound state of all target generator subsystems. Based on a mission scenario, the Executive downloads setup parameters to each of the target generator modules and to the scenario modeling system which controls the overall operation of the test. The Executive also formulates commands which coordinate the activities of the target generator and System Under Test (SUT) Simulator during self-test and calibration operations.

Director provides intuitive and consistent point and click control over all aspects of a testing cycle; that is, mission development, consistency checking, real-time monitoring of players and hardware, and self-test and calibration of subsystems. Director allows the user to select a working environment (or tool) with a single mouse click. All working environments provide on-line context-sensitive help. Director allows the user to quickly build scenarios, calibrate the target generator hardware, and set up tests and simulated missions. The visual technology provided with this test engineering environment lets the user "drag and drop" (that is, select an icon and drop it onto an appropriate form drop point) a wide assortment of objects like targets, jet engine modulation, clutter, calibration tables, radar cross section, etc.

Figure 1 illustrates the structure of a target generator scenario setup. The setup parameters shown in this hierarchical design range from hardware configuration and chamber setup to SUT setup and player setup. Defining a scenario could become very involved and time consuming; however,

with Director, a scenario can quickly be built using the familiar drag and drop feature. Previously constructed flight path, jet engine modulation, electronic countermeasure, and target characteristic objects are acquired from the library function, edited as necessary, and assembled to quickly create a test scenario. This easy-to-use scenario definition process allows multi-target scenarios to be built in minutes.

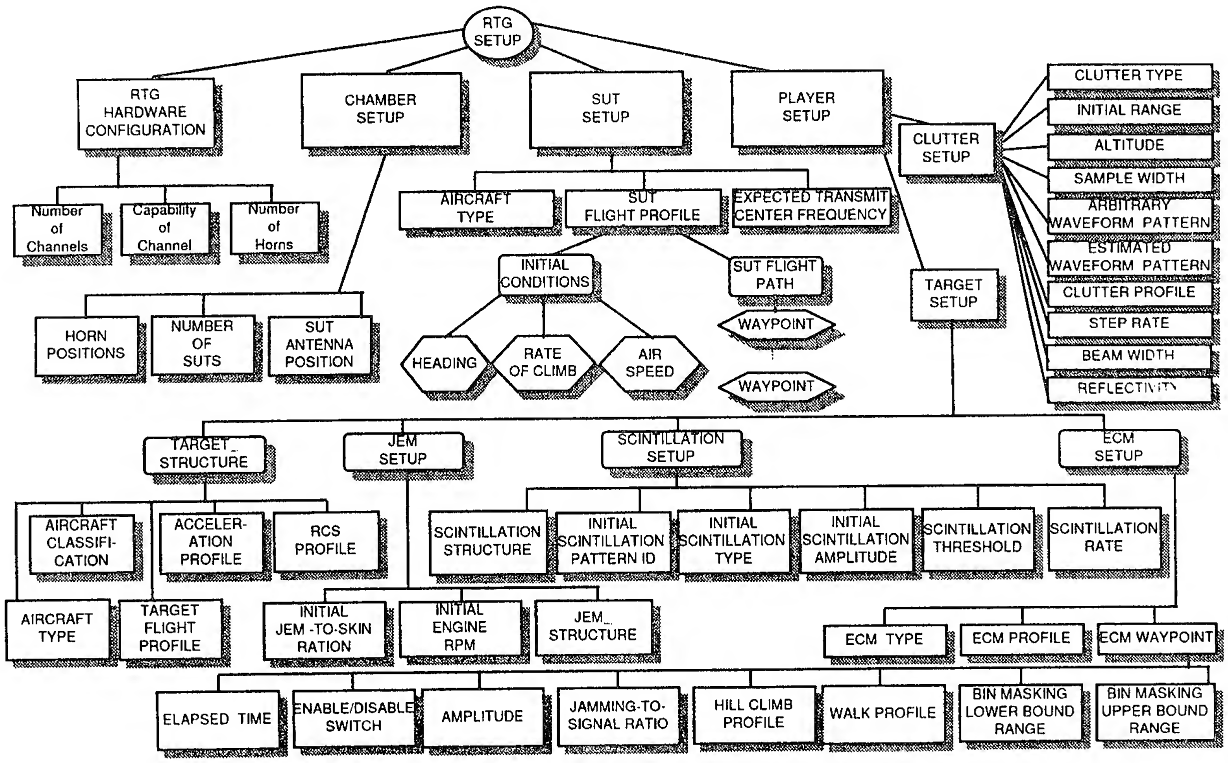


Figure 1 Scenario Setup

With the click of a button, the user can also display a powerful satellite view of a scenario in progress. The satellite view is also used to preview a scenario without having to receive or generate RF. By clicking on a player object that is displayed in a scenario satellite view, the user can observe or change the real-time parameters of that player; another click and the selected parameters are displayed in a strip chart. To set up a test, the user simply employs the mouse to select from the library a previously constructed and previewed scenario which contains all of the elements to set up a mission. Any setup function is just a mouse click away. For example, clicking on the download button downloads the scenario setup. The user is prompted when the download is complete and may then click on the start button or any other control button to further control the operation of the target generator.

Director is visually organized as groups of icons, each of which represents an environment or an object. Figure 2 illustrates the icons as they appear on the Director main menu bar. A brief explanation of the environments or tools which compose Director is addressed below in the order they might be used for setting up and executing a test.

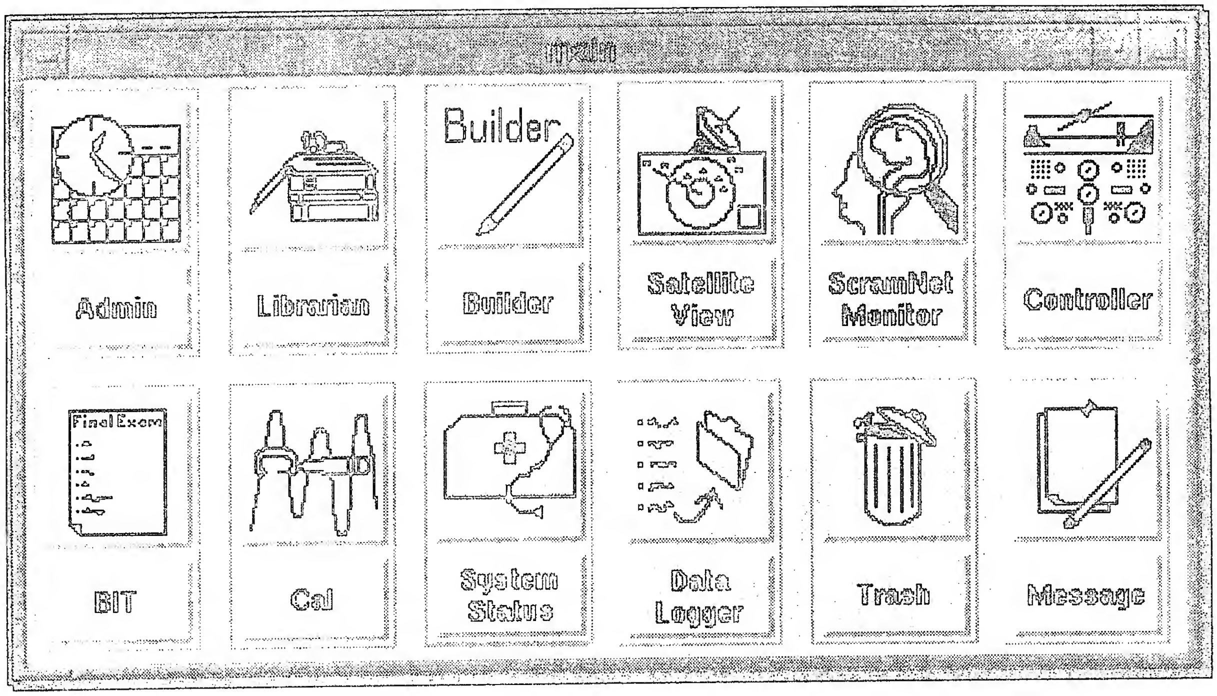


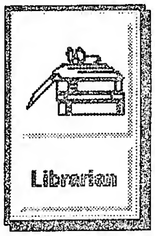
Figure 2 Main Menu

Administrator Tool

By utilizing the Administrator tool, each user is able to build a custom toolbox by selecting a set of environment icons presented on the target generator main menu. For example, a made-to-order maintenance toolbox might be composed of the following icons: BIT, CAL, SCRAMNet Monitor, and Maintenance. The user can also select which environments to expose (or display) upon system power up. For instance, the user may want the BIT tool and Status tool to be invoked and displayed each time the target generator is started.

After initially using the Administrator environment to custom-build a toolbox and configure the screen with the exposed environments desired, the user can save this screen configuration with a simple mouse click. This convenient feature provided by the Administrator gives the user complete control over which screens are exposed at start up. Each user can begin at power-up of the target generator with a customized environment, enabling all users to be more productive in their tailor-made environment.

Librarian Tool



The Librarian environment provides access to the target generator library which is a context-sensitive directory structure from which objects may be acquired via drag and drop. When selected from the library, these objects, represented by icons (as seen in Figure 3), may be dragged and dropped into other environments or may be dragged from other environments and dropped in the Librarian environment for storage. This tool conforms to the established interface consistency of Director by being fully integrated with all other environments. Another convenient feature of the Librarian tool is its use of filters to enable the user to quickly navigate with the familiar point and click technique. New objects may be checked into the library just as easily as they are checked out; the test engineer drags an icon from the definition form and drops it onto the Librarian tool icon.

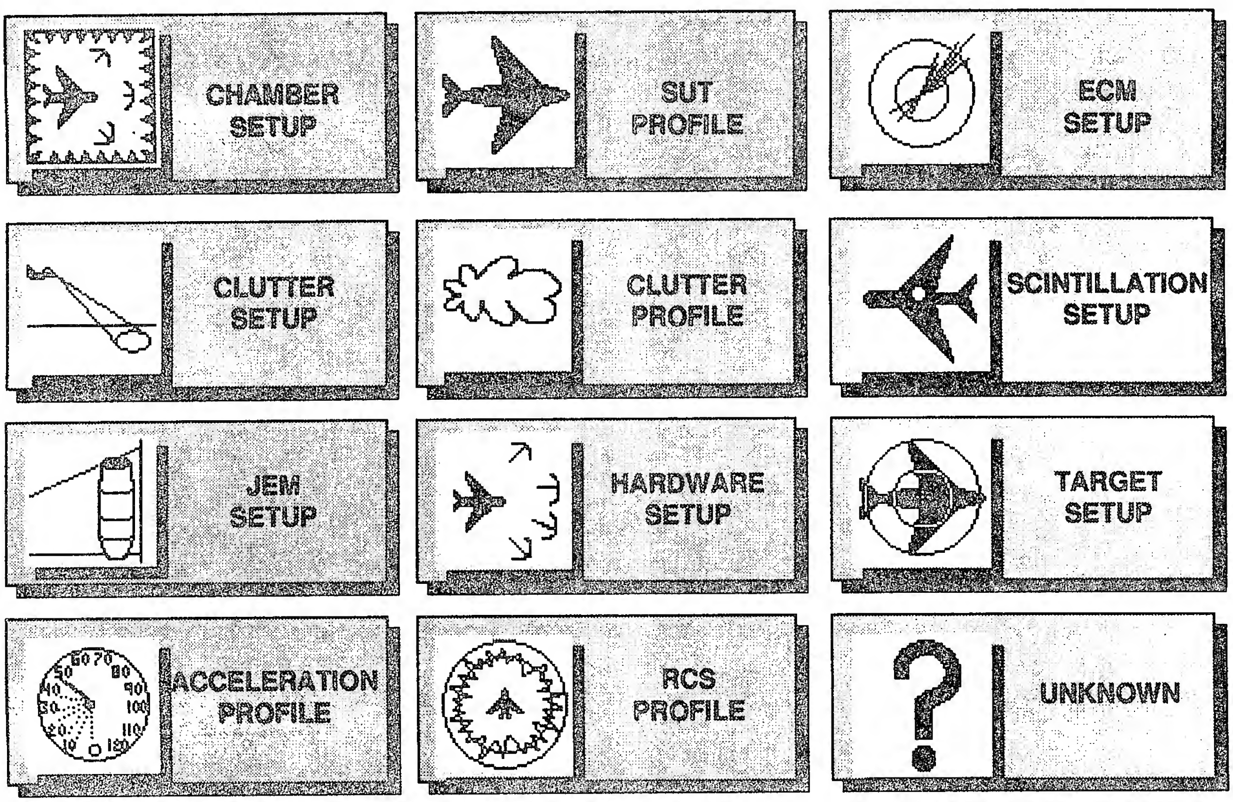
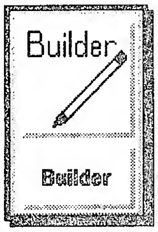


Figure 3 Standard Library Icons

Setup Builder Tool



Director enables the user to perform mission setup operations through the Setup Builder environment. This tool allows for the creation, modification, and storage of a scenario. Examples of some of the setup forms provided in this environment are shown in Figure 4.

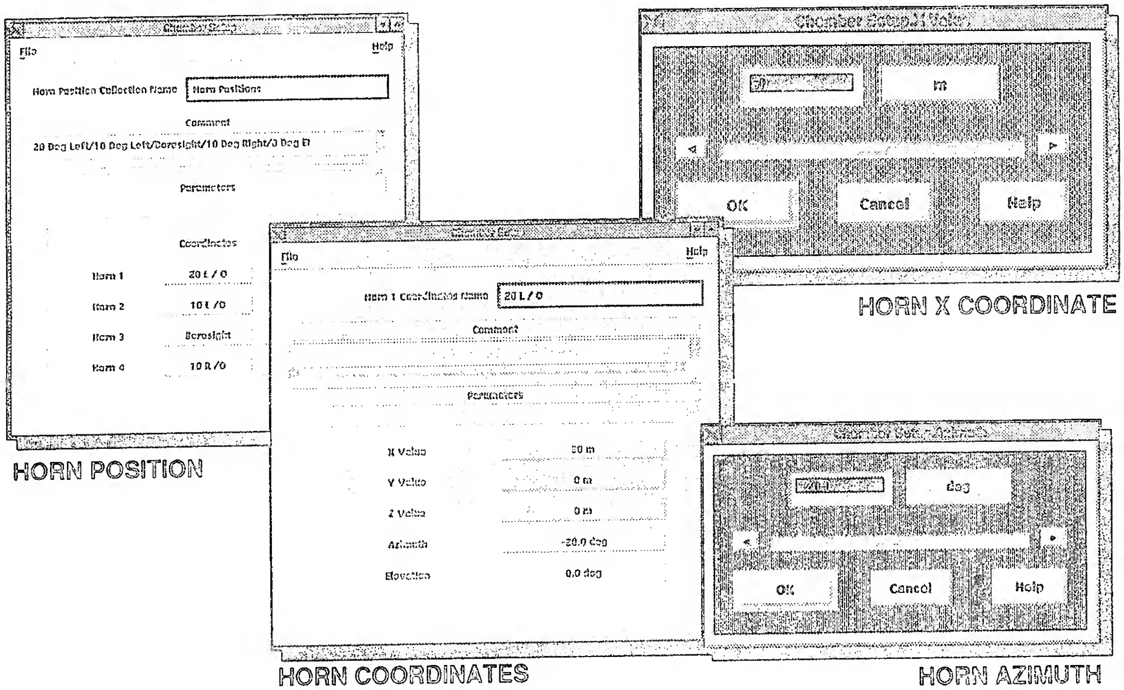


Figure 4 Setup Forms

A scenario is simply a mission, and the scenario components define the mission parameters. The mission setup operations include the capability to define target characteristics and mission flight paths. The user is also able to define the simulation objects required of the target generator. These objects include radar cross section, jet engine modulation, electronic countermeasures, and scintillation for each target, as well as main lobe clutter, side lobe clutter, and altitude line clutter.

In the Setup Builder environment, the user also has the capability to specify the test data to collect, to download data files and hardware configuration information to the target generator, and to preview a test scenario. Previewing a test scenario includes examining target and scenario data, and running and displaying target positions without receiving or generating RF. When the Setup Builder icon is selected, the user is initially presented with the main target generator setup form used to build a scenario. A scenario includes the hardware configuration setup, chamber setup, SUT setup, and player setups. (Please refer to Figure 1 for a diagrammatic description of the target generator scenario setup files.)

Separate setup forms are provided for each scenario object. Each object is connoted by an icon. Using a drag and drop mouse sequence, a test engineer can acquire, drag, and drop an object icon onto an appropriate form drop point. Figure 5 depicts an object icon being dragged from the librarian and dropped onto a Director builder setup form. Using this method repeatedly to perform mission setup operations, a scenario is built of new and reused objects.

Once a scenario has been composed, it may be checked by the scenario compiler and previewed on the scenario monitor prior to the scheduled mission to ensure that the parameters are realistic. Thus, multiple scenarios may be built and pretested in the convenience of an office with a desktop X-terminal.

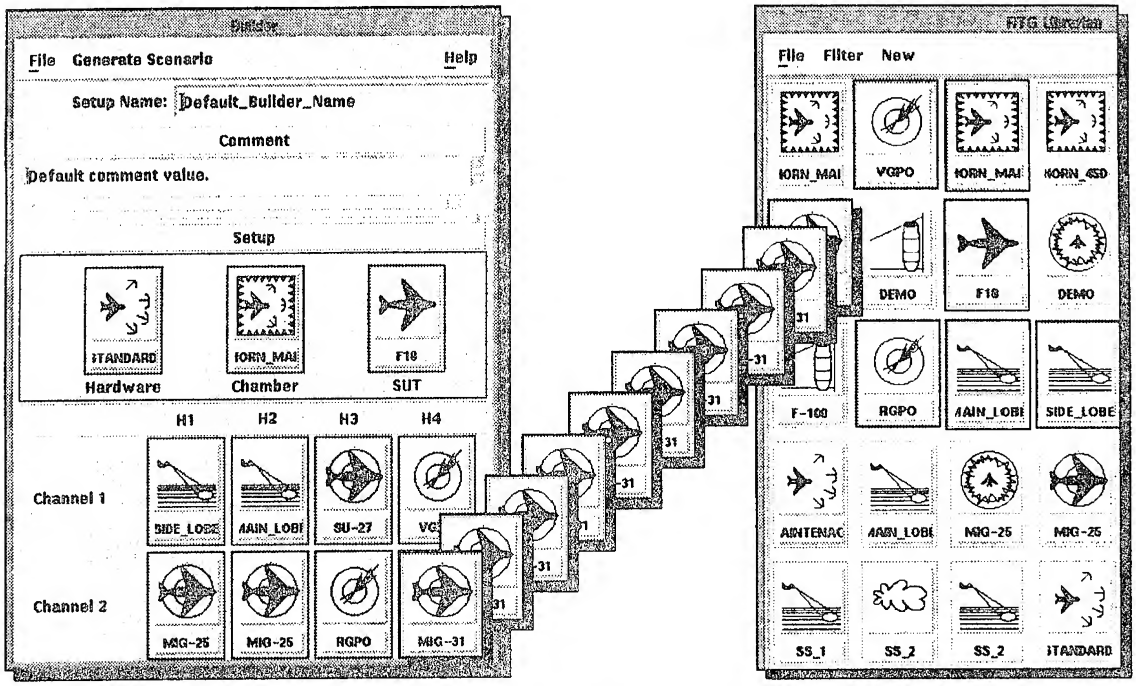


Figure 5 Drag and Drop Feature

SUT Simulator Tool

The SUT Simulator environment offers an intuitive, straightforward interface to a set of commercial test equipment which makes up the SUT Simulator. This environment provides the user with a convenient way to define waveforms and to adjust frequency, amplitude, and frequency offset. Arbitrary waveforms may be retrieved and played back while in this environment. True to form, the SUT Simulator tool is consistent with the other tools available in Director. Consequently, the user may store and retrieve any test created with a Director application. The SUT Simulator environment is a powerful testing and debugging tool which is directly accessible from the BIT and CAL environments.

Scenario Monitor

When in the Scenario Monitor environment, Director displays a satellite view of the mission players and their relationship to the SUT. While a scenario is in progress, Director displays and updates each player in near real time. The priorities for displaying data are driven by the user's ability to perceive change, which is the basis of the urgency to display data. The user may use a mouse to click on a player, which brings up a form displaying player parameters, such as range and range rate, again in near real time. The user may manually control a player's parameters using this same form by modifying parameters associated with this player. A strip chart display provides another view of a mission being executed. The user may choose the

players whose parameters are to be displayed in near-real-time strip charts as illlustrated in Figure 6.

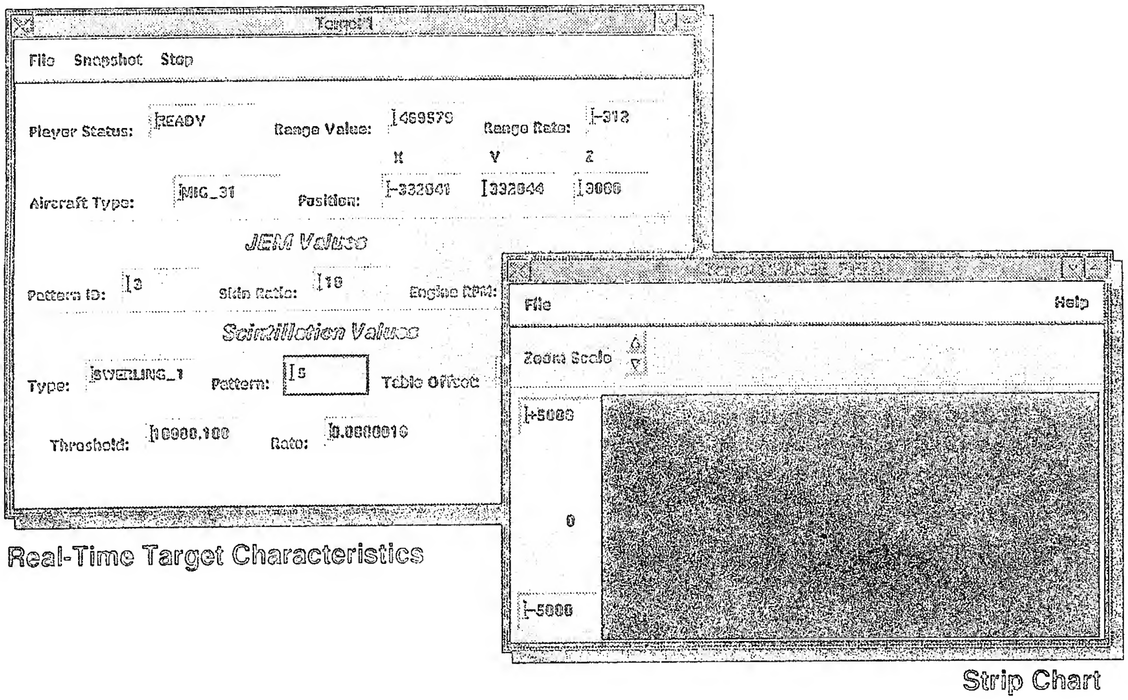
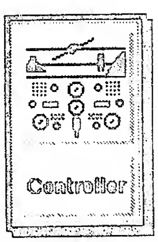


Figure 6 Real Time Displays

SCRAMNet Monitor Tool

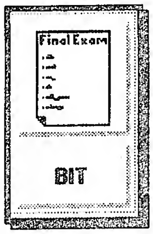
The SCRAMNet Monitor environment provides a debugging tool for target generator system data. The user enters an address offset and the size of the memory area to be displayed. The SCRAMNet Monitor displays these raw words (bit patterns) from the SCRAMNet in near real time. A strip chart display provides another view of data on the SCRAMNet. The user may choose one or more memory locations to be displayed in near-real-time strip charts.

Controller Tool



Scenario control is provided through the Controller environment. It allows the user to choose and download scenario setup data; enable, disable, and reset RF channels; start or stop one or all of the player models; and pause or continue a scenario once it has begun execution. In this environment, the user can execute and monitor stand-alone tests of a radar SUT. The target generator will receive RF from the SUT and generate and transmit target returns, clutter, and ECM in accordance with the defined test scenario and target characteristics. The Executive maintains the state and provides control for all RF modules and scenario models during test execution.

BIT Tool



The BIT environment enables the user to command the target generator to perform a list of built-in tests (BITs) and display the results in a Motif window. In the BIT environment, the user is presented with a list of tests that may be performed. Figure 7 shows an example of a target generator BIT environment from which the user can select the BIT (or BITs) to perform. Once a BIT has been initiated, this form monitors the progress of each test as it is executed, and a pass or fail indication is displayed at the completion of each test.

The Executive provides all required coordination with the SUT Simulator during a built-in test. The Maintenance environment can be accessed from the BIT tool, enabling the user to insert an argument command directly into an RF module memory address or control register.

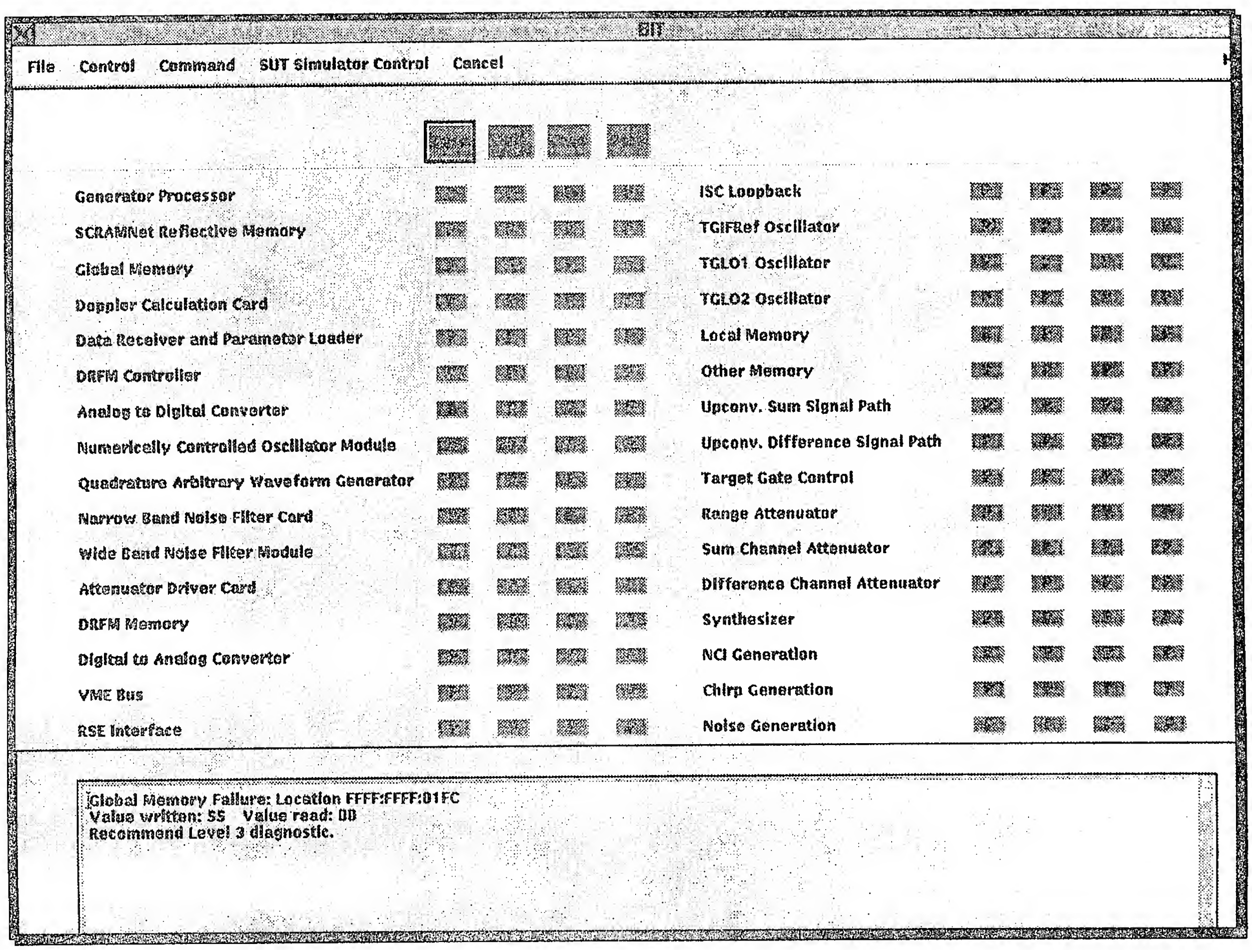
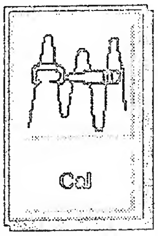


Figure 7 BIT Environment

CAL Tool



Running an end-to-end calibration of the target generator can be accomplished with the CAL Tool. The user is permitted to build and store calibration tables while in the CAL environment, as well as download previously generated calibration tables to the target generator.

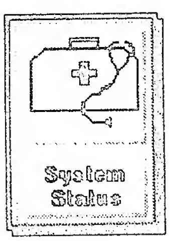
Through the CAL environment, Director presents the user with a list of components which can be calibrated. The user may then choose to calibrate a component or to use previous calibration data which has been stored. The Librarian tool has a list of calibration data files from which to select.

The Executive function of Director provides all required coordination with the SUT Simulator during calibration of the target generator. As in the BIT environment, the Maintenance environment can also be accessed from the CAL environment. This feature provides the user with a quick way to enter maintenance commands directly into an RF module memory address or control register.

Maintenance Tool

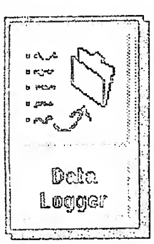
The Maintenance tool provides convenient memory and control register access to each RF module. When testing and debugging with Director, the user is not required to use a separate console to access the RF subsystem. The user can select the Maintenance icon from the main menu to invoke an environment that allows the user to directly access the RF module's memory and control registers by specifying the desired address and the command argument. At the push of a button, the command is initiated and the argument values are transferred to the selected location. As previously mentioned, Maintenance environment can also be invoked from the BIT and CAL environments.

Status Tool



In the Status environment, the current status of each target generator subsystem is displayed and updated in near real time. It is the responsibility of each subsystem to update its own status to its corresponding local SCRAMNet node. If the Status tool is iconified, a change in status will cause a color change in the icon notifying the user of a status change.

Data Logger Tool

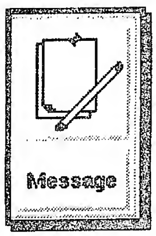


The Data Logger environment offers a channel to log real-time data to a file. The real-time data may come from an uninterpreted (or raw) data area of the SCRAMNet. An example of uninterpreted data is instrumentation data. This environment provides a form which allows the user to choose a file update rate, an address offset and length for raw data, or other parameters, such as the number of players in the case of real-time player state data. Director logs and time tags snapshots of the requested data at the requested rate.

Trash Can Tool

The Trash Can tool is provided by Director for the user to dispose of unwanted objects generated during target generator activities (i.e., configuring, testing, calibrating, logging). Objects are not actually deleted from the computer until the user empties the trash.

Message Tool



In this environment, a message window can be used by any subsystem to report errors, warnings, debugging information, etc. The user may filter messages, before they are displayed in the message window or stored in a message log. These messages are displayed and/or logged with a time tag and source. If the Message tool is iconified when a message is received, the icon will change color notifying the user that a message is waiting.

Summary

A common engineering environment is an important step in promoting commonality between the DoD test facilities. The workstations that execute Director can be used to control an RF stimulator, a millimeter wave stimulator, an infrared stimulator, or a CNI stimulator as long as these systems can interface to the SCRAMNet. This capability thus supports integrated operation of target generators in many spectra. A common test engineering environment, such as Director, also permits a high degree of synergy between testing in facilities such as the Benefield Anechoic Facility (BAF) and the Integration Facility for Avionics System Testing (IFAST). By using a common test engineering environment, test engineers at separate facilities can conduct tests in one facility and confirming tests in the other, simply by transferring the test scenarios files and repeating the test. Director represents the next generation of a user-friendly target generator engineering environment. Director provides insight into the test process and promises to make the test engineer more efficient.

ADAMS
Advanced Digital Avionics Methodology Schema
Automating the Test Process for Electronic Combat System OFP Software

Michael J. Willis
Georgia Tech Research Institute
Atlanta, Georgia 30332

Abstract

Complex Electronic Combat (EC) equipment has become increasingly software intensive with each advancement of technology. As the programmable features and available memory for operational software and data tables have increased, the requirements have increased for comprehensive testing to validate these EC systems. Simultaneously, field testing of the Operational Flight Program (OFP) software and associated Emitter Identification (EID) data bases has also become increasingly complex, costly, and time consuming. A requirement exists to develop an affordable and timely methodology to improve the test methods used to validate these systems before field testing or operational use.

WL/AAAF's ADAMS develops, prototypes, and demonstrates a methodology to perform more rigorous and timely RF testing of EC system/subsystem OFP software. Through efficient utilization of the data busses employed in modern avionics equipment such as a Radar Warning Receiver, this schema augments the current labor intensive testing conducted on dedicated RF test simulators. By providing virtual test points within the OFP, automated tracking of the signals used to stimulate the Unit Under Test (UUT) and the UUT's output performance data may be achieved.

The Georgia Tech Research Institute (GTRI) has developed PC-based ADAMS software to control the Advanced Standard Threat Generator (ASTG) currently used by the Warner Robins Air Logistics Center. The ASTG is used by WRALC to simulate complex threat scenarios for test and validation of RWR and other EC system OFP software. Prior to the development of the ADAMS software, control of the ASTG and scoring of RWR performance has been a time consuming manual process. To evaluate the feasibility of ADAMS, the AN/ALR-69 Class IV Upgrade RWR was selected as the initial test vehicle. By monitoring the RWR outputs on the MIL-STD-1553 data bus and programming the ASTG for a sequence of single site emitters, the ADAMS software improved test times by a factor of two, and provided automatic scoring, documentation and reports for the test operators. Additionally, ADAMS allows testing of the EC system OFPs to be performed around-the-clock, and does not require continuous operator attendance.

Although initially intended as a prototype system, the ADAMS concept was immediately implemented by WRALC as part of the Block Cycle Update process for the ALR-69 Class IV OFP software. Current improvements to ADAMS include modifying the software to allow side-by-side testing of two RWRs, and extending the ADAMS concept to scoring of

integrated EW systems.

Introduction

Complex Electronic Combat (EC) equipment has become increasingly software intensive with each advancement of technology. As the programmable features and available memory for operational software and data tables have increased, the requirements have increased for comprehensive testing to validate these EC systems. Simultaneously, field testing of the Operational Flight Program (OFP) software and associated Emitter Identification (EID) data bases has also become increasingly complex, costly, and time consuming. As a result, the requirements for monetary and human resources needed to properly validate software updates for EC systems have increased. These escalating resource requirements have occurred at the same time as significant reductions in the size of many government programs. In an effort to improve the timeliness of the validation process for EC system software updates while maintaining or improving the quality of the validation effort, Wright Laboratory conceived the Advanced Digital Avionics Methodology Schema (ADAMS) in cooperation with test personnel at the Warner Robins Air Logistics Center. The Georgia Tech Research Institute (GTRI) was selected by WL and WRALC to develop the ADAMS concept to a prototype level, using the AN/ALR-69 Radar Warning Receiver as the initial test vehicle.

Selection of the AN/ALR-69(V) RWR as an appropriate test vehicle was based on several factors:

1. The system has recently completed a major Reliability and Maintainability Upgrade. The resulting RWR, known as the ALR-69 Class IV Upgrade, features a significant increase in memory and computing power relative to the previous version. As such, the Class IV Upgrade is an excellent example of the type of system described above, possessing increased processing and data storage capabilities.

2. The ALR-69 is currently the most widely fielded RWR in the Air Force, Air Force Reserve, and Air National Guard fleets. As a result, improvements in the test validation process will have a widespread benefit.

3. GTRI has been involved in the design, development, test, and analysis of software and hardware for the ALR-69 for over fifteen years.

4. The AN/ALR-69 Class IV Upgrade, with the inclusion of the MIL-STD-1553 communications bus and associated OFP software, contains the interface necessary to support automated real time measurement of system performance data.

ADAMS Requirements

Improving the timeliness of any effort typically requires some level of efficiency

improvement. In the case of software validation, these improvements may readily take the form of process automation. For the AN/ALR-69, the software validation process has required the operator to manually control the ALM-234 Advanced Standard Threat Generator while simultaneously monitoring (visually) threat symbols and system data on the RWR display.

In order to quantify the efficiency improvements of ADAMS, the program requirements shown in Table 1 were established.

Table 1 - Requirements for ADAMS

Requirement	Description
1.	Increase operational readiness of the AN/ALR-69
2.	Provide for more robust testing of OFP
3.	Demonstrate the concept of automated digital testing and scoring
4.	Increase test and scoring reproducibility
5.	Automate the One-on-One testing
6.	No impacts to the Advance Standard Threat Generator (ASTG AN/ALM-234)
7.	No impacts to the AN/ALR-69

These requirements imposed significant, but necessary, restrictions. In particular, the requirements to not modify the ALM-234 or the ALR-69 were of primary importance. The ASTG is used extensively as a test stimulus standard for system validation and evaluation at WRALC. Modifying this system would result in the need to revalidate the test system itself - a task which was not within the scope of the program. The ALR-69 is a fielded, operational system. To modify the system in order to complete the validation process would be ill-advised and inappropriate. Therefore, it was necessary to determine what performance data was available from the RWR and take advantage of the data through capturing in software and automating the analysis.

<u>Approach</u>

Achievement of the requirements set forth above was accomplished by development of Windows-based PC software with the following significant elements:

1. Test, Scoring, and Reporting Software (TSAR) which analyzes the UUT's performance data received via a MIL-STD-1553 serial data bus.

2. Terminal Emulation Software to provide the automation link between TSAR and the ALM-234.

3. Formatting and storage of received performance data as Log Files and Score Files to document the test process.

The block diagram of the ADAMS system is shown in Figure 1.

ALR-69 Performance Data

The AN/ALR-69 Class IV Upgrade RWR features inclusion of a MIL-STD-1553 serial communications data bus. This bus is now a standard on many EC systems, both fielded and in development. The ALR-69 Class IV OFP software transmits the majority of the

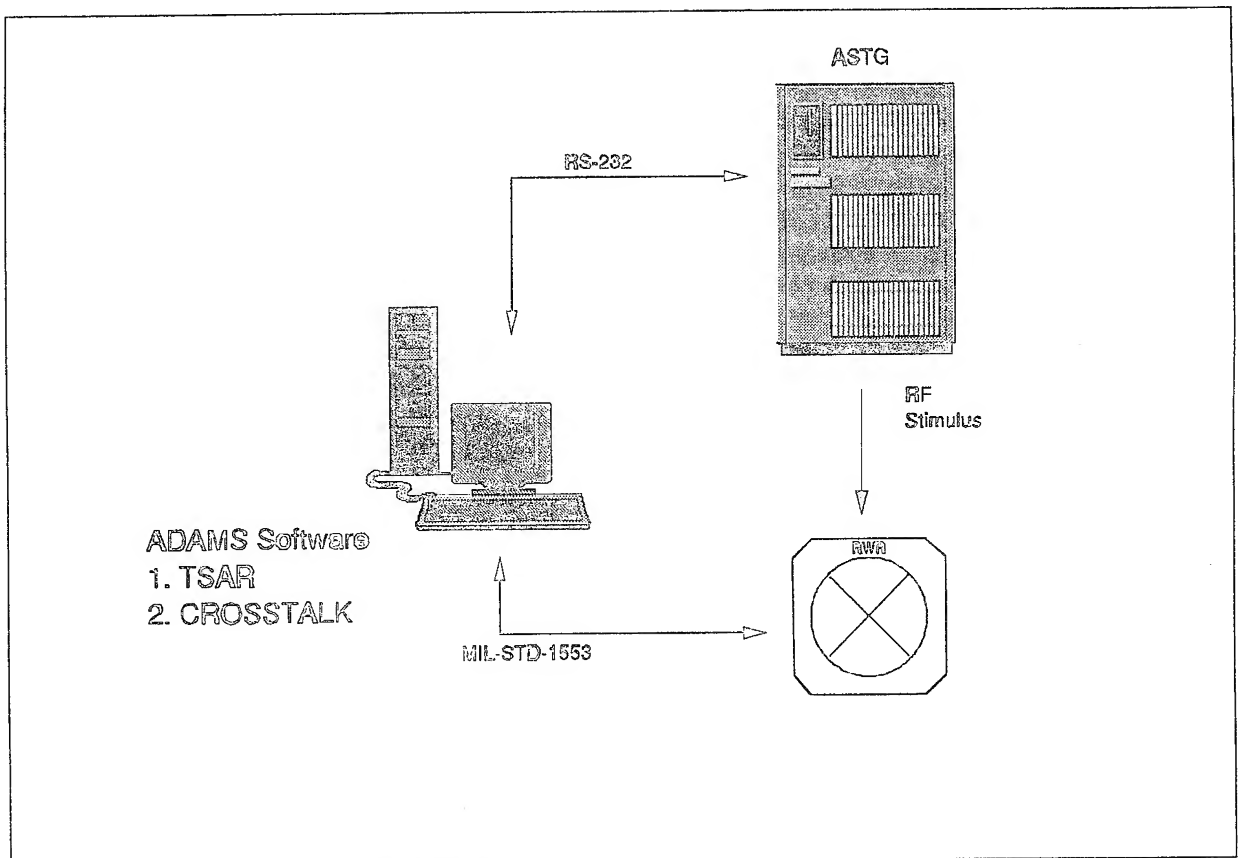


Figure 1 - ADAMS Configuration

performance data for the RWR on this bus when commanded. This feature readily supports the requirement for monitoring the UUT without modifying the system. Performance data available via the '1553 bus includes the following:

1. Input Buffers - The Input Buffers contain raw data on received pulses. Data includes measurement of pulse width, relative power, angle of arrival, time tags, blanking (if present), video pulse-on-pulse conditions, and band sorting.

2. Emitter Track Files - The ETFs contain processed data from the Input Buffers. ETF data identifies detected emitter types (T Types), relative power, angle of arrival, Missile Launch, and Missile Activity data for each pulse train.

3. Display Files - The Display Files contain processed data from the Emitter Track Files. For example, ETF entries may appear for brief moments. Before inclusion in the Display File, minimum detection times must be satisfied.

As a result of the availability of extensive performance data, the ALR-69 Class-IV Upgrade is able to support the automated test process without modification of system hardware or software. Availability of ETF data allows more detailed analysis of system performance, when compared to the previous process of visually monitoring the RWR display.

ASTG Control

The ASTG is normally controlled with a VT-320 serial terminal connected to the RF signal generator via an RS-232 communications link. The ADAMS system uses a commercial software package, CrossTalk for Windows, to emulate the VT-320 serial terminal. CrossTalk for Windows allows other Windows applications to control its input. In the ADAMS program, the Test Scoring and Reporting (TSAR) software controls CrossTalk to emulate the operator input. The ASTG is controlled in this manner without any need for modifying its software or hardware.

Log Files and Score Files

TSAR outputs from ADAMS are stored to disk as Log Files and Score Files. The Log Files provide a means of post analysis of the system data, while the Score File provides data for relative performance comparisons. Examples of a typical Log File entry and Score File entry are shown in Tables 2 and 3 respectively.

Table 2 - Example Log File for ADAMS System

Site demo2.sit

Setup

Comments:
Angle: Using Default
Expected Symbology: T Type 0012, Missile Activity, Missile Launch, Uncorellated Guidance
RWR Mode: Normal
ICU:

ASTG Files: File	Initial Power (dBm)	Final Power (dBm)	Function Generator	Pulse Generator	Priority	Sync Delay
E2B2_B.CL	Default		1	1	1	0
E2B4_1ML.CL	Default		2	1	1	0

Test Repetition Interval: 20 seconds
Number of Test Repetitions: 4

Iter	Serial	TTYPE	Time First	Time Corr.	Time AgeOut	Angle Low	Angle Avg	Angle High	ML O	ML C	MA O	MA C	UCG O	UCG C	Type History							
1	373	0012	1.5	1.5	22.6	347	347	347	1	1	1	1	1	1	0017	0000	0000	0000	0000	0000	0000	0000
2	377	0012	1.4	1.4	22.4	347	347	347	1	1	1	1	1	1	0017	0000	0000	0000	0000	0000	0000	0000
3	382	0012	1.5	1.5	22.3	346	347	347	1	1	1	1	1	1	0017	0000	0000	0000	0000	0000	0000	0000
4	385	0012	1.6	1.6	22.5	347	347	347	1	1	1	1	1	1	0017	0000	0000	0000	0000	0000	0000	0000

Table 3 - Example Score File for ADAMS System

Site demo2.sit

Setup

Comments:
Angle: Using Default
Expected Symbology: T Type 0012, Missile Activity, Missile Launch, Uncorellated Guidance
RWR Mode: Normal
ICU:

ASTG Files: File	Initial Power (dBm)	Final Power (dBm)	Function Generator	Pulse Generator	Priority	Sync Delay
E2B2_B.CL	Default		1	1	1	0
E2B4_1ML.CL	Default		2	1	1	0

Test Repetition Interval: 20 seconds
Number of Test Repetitions: 4

Power Test Number	Angle (degrees) Low	Angle Avg	Angle High	Display Time (seconds) Low	Display Time Avg	Display Time High	Correct ID Time (seconds) Low	Correct ID Time Avg	Correct ID Time High	Age Out Time (seconds) Low	Age Out Time Avg	Age Out Time High	Number of IDs Correct	Number of IDs InCorrect
0	346	347	347	1.4	1.5	1.6	1.4	1.5	1.6	22.3	22.4	22.6	4	0

Automation of One-on-One Testing

Validation of software updates for Radar Warning Receivers typically is initiated in a laboratory environment where signals to be encountered by the system are simulated by the ASTG. These signals are simulated individually and statically. That is, each simulated signal is applied to the system under test in a manner which replicates static distance and angle. A "signal" may consist of multiple RF waveforms if the simulated system employs such waveform generation. A typical example of such a radar system might include a Search RF waveform, a Track RF waveform, and a Missile Guidance RF waveform. The validation sequence includes application of the signal to the RWR, toggling the mode of the RWR to initiate a new detection sequence, and measuring the time required by the RWR to identify the applied signal. Finally, the signal is deactivated by the ASTG, and the time required for the associated symbol to be removed from the RWR display is measured. This is the system "age-out" time for the particular signal. The software validation process typically involves the application of dozens of signals to the system under test. It is therefore quite time consuming, if undertaken manually.

Previous versions of the ALR-69 did not include the MIL-STD-1553 data bus. Digital output data was limited, and performance evaluation was typically accomplished by visual monitoring of the RWR display. The evaluation process required two operators. One operator would control the ASTG, activating static emitters according to a defined test plan. The second operator would monitor the RWR display, recording measures of performance such as the time required for threat identification, identification ambiguities, angle of detection, and age-out times. Recording of performance data occurred through written notes.

Automation of the one-on-one test process is accomplished, using the ADAMS software, as follows:

1. Develop a script file for execution by the ADAMS controller. The script file contains necessary commands to program the ALM-234 for each signal. Use of scripts supports unattended data collection, since the operator can develop scripts to generate many signals, in sequence, and then allow the test to run automatically. The script file is written by the ADAMS software as the operator enters ASTG programming information into Windows Dialog Boxes. Programming information is requested in the same manner operators are accustomed to seeing when controlling the ASTG directly from a VT-320 terminal, thereby minimizing the need for retraining to successfully use the ADAMS software.

2. For each signal, program the ASTG from the script file. Then toggle the mode of the RWR via the MIL-STD-1553 data bus to initialize the system under test. This step also aids in determining the system identification time, since the primary undefined latency is the time delay of '1553 bus signal transmission. The ALR-69 Class IV Upgrade RWR supports system control via the '1553 data bus as well as

from the system cockpit control panel.

3. Receive ETF data from the system under test over the '1553 data bus. From this data, determine the detected angle and Emitter Identification.

4. Command the ASTG to remove the applied signal. Then measure the time when the RWR removes the detected emitter from the Emitter Track File.

5. Repeat steps 1-4 for each applied signal.

Results

Evaluation of the utility of the ADAMS prototype was accomplished by comparing the times required to complete a series of one-on-one tests manually and under the control of the ADAMS software.

This test evaluation revealed several items of significance:

1. For the test sequence specified, the time required to complete the test program was reduced by approximately 50% when the ADAMS software was employed.

2. Test repeatability was demonstrated. Operator errors that resulted in ambiguous data collection during the manual test sequence were eliminated when the tests were controlled by the ADAMS software.

3. Logging the ETF data allowed operators to detect subtle problems in the system software. Deficiencies were noted, debugged, and resolved in a fraction of the time typically required using previous test methods. System operational readiness was increased as a result.

Future Plans

The results of the ADAMS demonstration were of such significance that the software was immediately implemented as part of the Block Cycle Update process for IV&V testing of the ALR-69 Class IV system. In addition to the formal validation process, the program has been used for troubleshooting of the Operational Flight Programs (OFPs). This capability exists because of the extensive availability of performance data on the '1553 data bus. Because of this data availability, operators could determine the source of poor system responses to particular signals, and implement OFP improvements immediately. With the use of scripts and system automation, data consistency was improved. Thus, test operators were able to determine with greater confidence when a particular deficiency had been resolved.

The initial ADAMS prototype was developed for evaluation of single systems. A natural extension of the ADAMS concept is to modify the software to support side-by-side testing of identical hardware systems programmed with different OFP versions. This configuration

could support evaluation and scoring of existing OFPs and OFPs containing potential operational improvements, as shown in Figure 2. As illustrated in the figure, both RWRs are stimulated from a common source (the ALM-234 in this case). The ADAMS software must monitor two MIL-STD-1553 data busses simultaneously to capture system data. This modification is required because the two systems will reside at the same bus address.

GTRI is currently implementing these capabilities as part of the ADAMS Phase II program effort. Demonstrations of the side-by-side capability are planned for the latter part of 1995.

The ADAMS concept is not limited to evaluation of RWRs, nor is it restricted to control of the ASTG. Evaluation of other types of EC systems requires specification of appropriate performance parameters, and availability of data on the '1553 data bus. In addition to evaluation of individual EC systems, automated scoring of integrated EW systems is possible. With the inclusion of MIL-STD-1553 communications capability in many systems, work is now progressing in the area of integration of the these systems to provide improved operational capabilities for pilots and aircrews. For example, GTRI has demonstrated that

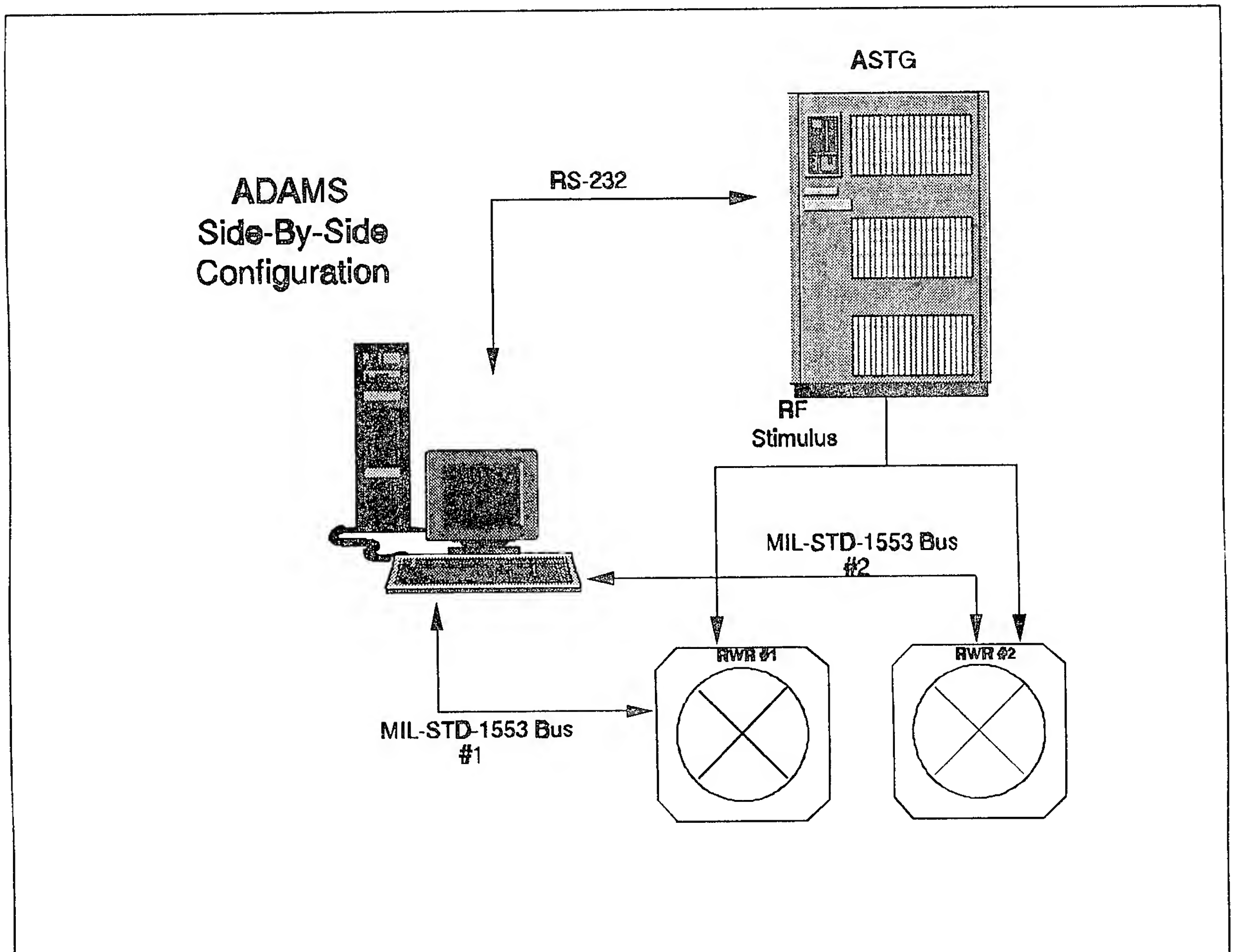


Figure 2 - Side By Side Equipment Configuration

improved operational capabilities are achievable when the AN/ALQ-131 electronic countermeasures system is integrated with the ALR-69 RWR via the '1553 bus. Historically, emissions from the ALQ-131 have been detected by the ALR-69, and the result has been false symbols on the RWR display. This phenomenon has been minimized as a result of recent software modifications. Investigation of additional integration capabilities, and scoring of the integrated systems, is a future goal of the ADAMS effort.

Since control of the RF stimulus equipment simply requires an appropriate communications interface, applying the ADAMS concept to systems other than the ALM-234 is straightforward. In cooperation with other contractors, future improvements to ADAMS will include control of the new Low Cost Interactive Stimulus Test Station (LISTS).

LOW COST INTERACTIVE STIMULI-GENERATING TEST STATION (LISTS)

Ronald G. Loeliger
Jerry L. Mulder
Hughes Aircraft Company

INTRODUCTION

Reprogramming Electronic Combat (EC) equipment Operational Flight Programs (OFPs) is a labor intensive task that relies heavily on skilled engineering and software verification and validation personnel. The reprogramming environment is often unique from system to system and even differs among the field and operational sites for any given system. The operational parameters describing the modes, waveforms and characteristics of the threat RF signals are contained in a variety of data bases, none of which is tailored to the reprogramming task. The simulation equipment that generates the microwave signals used to stimulate the reprogrammed equipment is generally expensive, proprietary in nature, complex, and non-portable. The testing and test data analysis software packages used in conjunction with the reprogramming activities are not an integrated, user friendly environment. Reprogramming EC equipment is therefore highly dependent on the skills of the personnel performing the task; personnel who are in shorter supply due to the current downsizing of the military.

To alleviate the demands on the reprogramming personnel, WL/AAAF has initiated a three pronged attack on the EC reprogramming problem using three interrelated programs. The Computer Assisted Test Development and Reporting System (CATDARS) program is formulating and developing a software based tool environment for exploiting and integrating system testing knowledge associated with the threat environment. The A Digital Avionics Methodology Schema (ADAMS) project is developing a methodology that performs more rigorous, realistic and timely testing of Operational Flight Program (OFP) software by utilizing data captured from the data buses of the Unit Under Test (UUT) to score the reprogramming effectiveness. The LISTS project is developing a low-cost, portable, high fidelity microwave threat environment for stimulating the EC equipment. While each project is currently in the proof of concept stage, the programs are tightly integrated to support the eventual synthesis into a common EC reprogramming environment.

The LISTS project, a task under the Wright Laboratory (WL/AAAF) Readiness of Embedded Computer Systems (RECS) program, was undertaken to develop a modern prototype RF simulator of the current and emerging threat systems to augment the existing threat simulators. The first part of this paper discusses the trade-off and design studies involved in the initial development of the LISTS system requirements and details how the system architecture will be developed. The second part of the paper describes the present planned system architecture and how it will be implemented in the LISTS prototype. Finally the integration of the LISTS with the ADAMS and CATDARS technologies will be explored.

LISTS TRADE-OFF AND DESIGN STUDIES

The goal of the LISTS prototype effort is to produce a low cost, versatile prototype simulator capable of serving a wide range of needs. Accomplishing this will require a survey of the existing emitters, the

emitters which are anticipated in the future, and the test environments at various organizations which are testing EC equipment. These surveys will produce a set of global requirements which the LISTS architecture must attempt to satisfy. Those requirements which will incur significant cost increases and/or are needed to satisfy only a small percentage of users will be isolated and considered individually. Trade-off studies will be performed as necessary to determine whether a specific requirement will be included in the LISTS architecture design.

The requirements which come out of the above effort will be used as the baseline for the LISTS architectural design. A further subset of those requirements will be chosen for actual implementation in the LISTS prototype system. The intention is to produce an architecture capable of meeting as many of the global requirements as possible and a prototype system which demonstrates this capability on the subset of requirements which can be achieved within the prototype budget. Emphasis will be placed on using commercial standard buses with commercial-off-the-shelf (COTS) hardware wherever possible.

LISTS SYSTEM ARCHITECTURE

The LISTS system architecture will directly attack the goals of low cost, easy reprogramming, and portability by employing a modular approach to both the hardware and software design. By creating a set of basic building blocks in both hardware and software, the LISTS prototype effort will produce an architecture which will be valid for a small field portable unit or a lab based unit capable of simulating a large set of complex threats. To attempt to make the LISTS be all things to all people would be an unrealistic goal, however, if the architecture can at least come close to achieving this, then the LISTS will be a viable simulation platform for most future EC test requirements by adding or modifying building blocks to satisfy the new requirements. Additionally, new technologies will be explored for low cost implementation of components within the LISTS RF chain.

Electronic Architecture

The LISTS system will be comprised of three main parts, the Chassis, the Controller, and the RF Module. The prototype Chassis will be a rack mount enclosure which will support plug-in RF Modules with electrical power and cooling. It will also provide RF circuitry to combine the outputs of the various modules into a single set of outputs for the EC unit under test. The Controller will be a COTS computer system which will communicate with the modules in the chassis via a COTS interface protocol. The Controller will communicate with other systems in the test environment via Ethernet. The RF Module will be a self-contained emitter simulator with an embedded processor and digital and RF circuitry. Figure 1 shows a block diagram of this architecture.

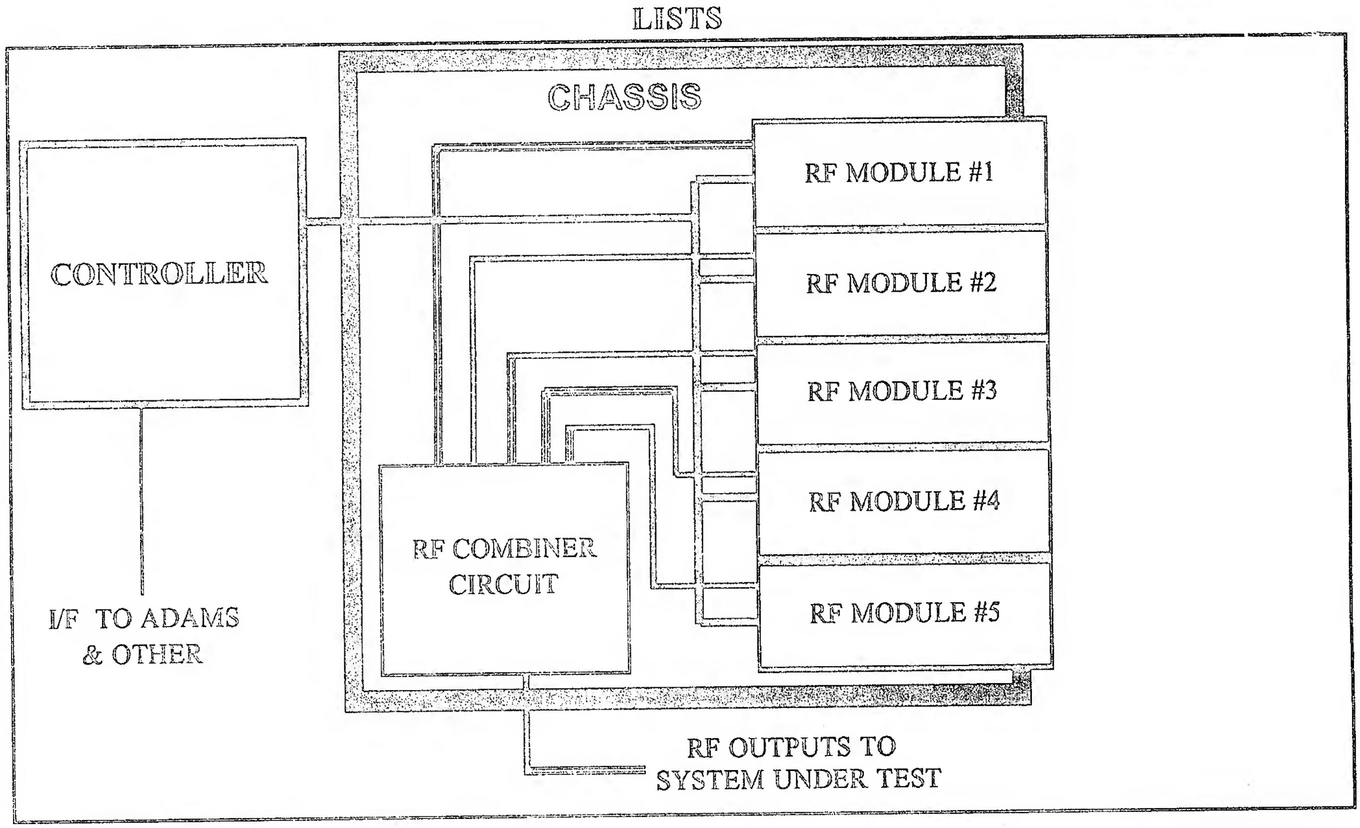


Figure 1: LISTS Block Diagram

Mechanical Architecture

The design of the mechanical architecture will be a challenging part of the LISTS prototype effort. The main focus will be on the design of the Chassis Backplane and the plug-in RF Modules.

The mechanical design of the RF Module must support the digital and RF inputs and outputs necessary to meet the LISTS architectural requirements. The module size must also be directly scaleable in the sense that as different types of RF Modules are developed to meet varying requirements, the module footprint can expand or contract as needed to house the appropriate hardware.

The Chassis Backplane must support both digital multipin connectors and RF connectors. Most importantly, it must support the variable RF Module footprint.

Software Architecture

The LISTS software will be divided into at least two major Computer Software Configuration Items (CSCIs); the Controller CSCI, and the RF Module CSCI. The partitioning of tasks between these two will place as much of the real-time processing load as possible into the RF Module CSCI. The Controller CSCI will handle the graphical user interface, assignment and coordination of emitter simulations to RF Modules, and any external interfaces to other system in the test environment.

LISTS

The RF Module CSCI will have to be able to identify the configuration of the RF Module in which it is running and call specialized routines as appropriate to control that modules specific hardware and make use of its full capabilities.

The Controller will be the host for the LISTS software development, to the extent practical. The software for the embedded processor in the RF Module will possibly have to be implemented in assembly language due to speed considerations. The plan will be to use a cross-assembler for the embedded processor, hosted on the Controller. If one is not available, an additional development workstation may be required to support software development. A large portion of the Controller software will be COTS software. The operating system, development tools and even portions of the operational applications, e.g., the data base management software, will be COTS.

PROPOSED IMPLEMENTATION

RF Module

The RF Module will itself be broken down into sub-blocks. These sub-blocks will include but are not limited to;

- the processor block which will be an embedded CPU that will handle the real-time task of controlling the other sub-blocks to generate pulses for the emitter simulation. This embedded CPU could be packaged on a custom printed circuit board (PCB) along with the other digital circuitry or it may be a COTS processor board on VME or VXI bus.
- the digital timing and control block which will contain specialized circuitry to form the pulse train, phase modulation, amplitude modulation, and amplitude control of the RF sub-blocks. This will be packaged on a custom PCB.
- the RF source block which will be made up of some type of continuous wave (CW) RF frequency source such as a voltage controlled oscillator (VCO), digitally tuned oscillator (DTO), or synthesizer.
- the global modulation block which will apply amplitude and phase modulation to the single output of the RF source block.
- the channelization block which will split the single RF frequency coming from the RF modulation block into multiple channels as the test requirements dictate.
- the channel modulation block which will apply individual amplitude and phase modulation to the RF channels formed in the RF channelization block. This would likely be the final sub-block in the RF Module and the outputs formed would be the RF Module outputs.

- the calibration and self test block which will contain circuitry such as RF switches, RF detectors, etc. that will receive sample signals from the other RF sub-blocks in order to allow the embedded processor to perform operational tests as well as diagnostic tests.

Figure 2 shows the block diagram for the proposed RF Module.

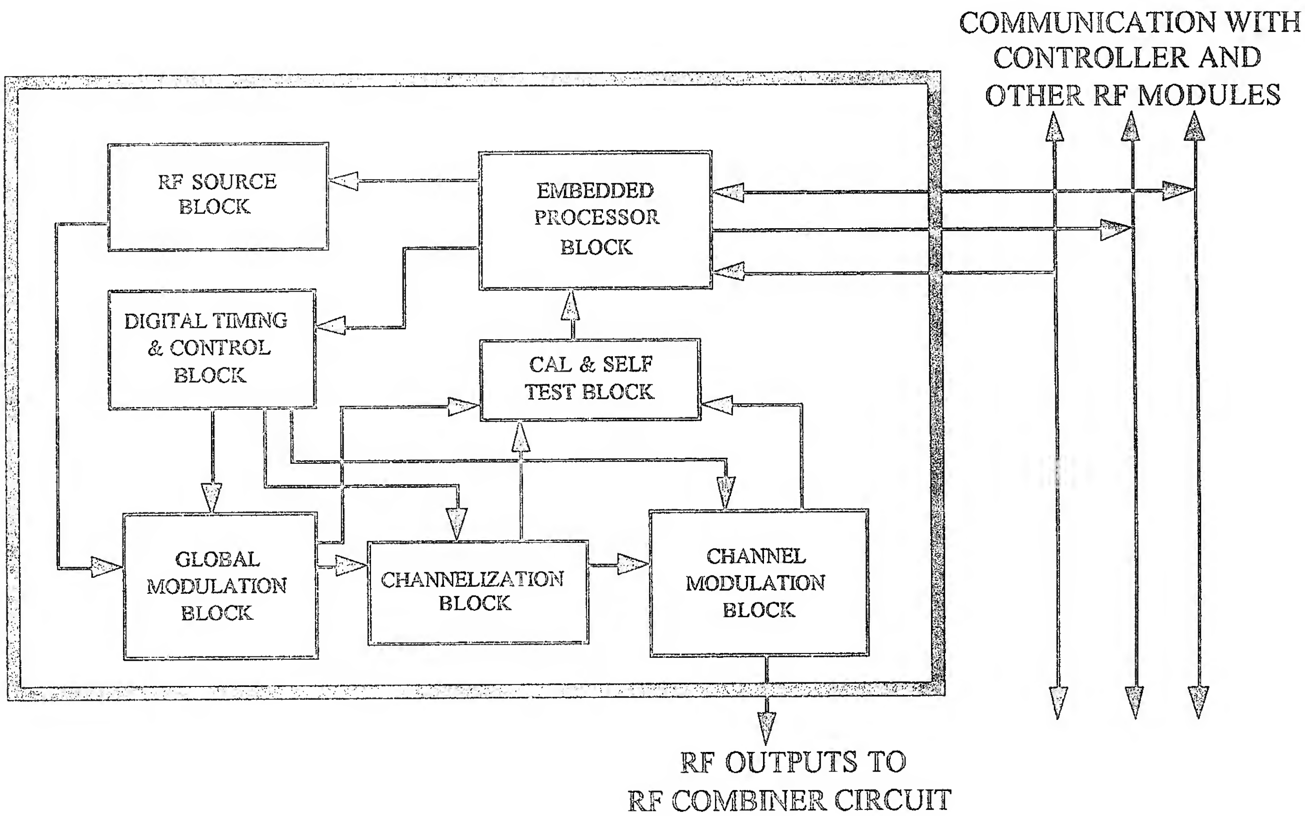


Figure 2: RF Module Block Diagram

The design of each of these sub-blocks will make use of a common control bus within the RF Module itself, which will be sourced by the processor block. Along with this common control bus there will be dedicated digital signal paths for synchronization signals, clocks, etc. The idea here is that each class of sub-block could have multiple types which would be electrically interchangeable with each other. For example, one type of RF source sub-block could be built around a 2-18 GHz DTO, another could be built around a relatively slow tuning 2-6 GHz VCO, while a third could employ a synthesized source with sub-hertz frequency resolution. Similarly, one type RF channelization sub-block could produce 5 channels of RF while another type produced 8 channels. The other sub-blocks would have multiple types, each with different capabilities. Additionally, the architecture will make it possible to incorporate as much or as little functionality into each sub-block as is needed. If a particular class of emitter simulation requires extensive harmonic suppression, these harmonic filters could be incorporated into the appropriate RF sub-block of that RF Module. Likewise, if a particular RF Module does not need to provide AOA phase modulation to the individual channels, the RF channel modulation sub-block for that RF Module would not need to have phase shifters in it. The design goal is not to have the functionality

and capability of the RF Module limited by the underlying architecture. The only limitation should be the volume available in the RF Module for mounting components.

Chassis

The LISTS prototype Chassis design must accommodate the plug-in RF Modules and will be complicated by the possible multiple sizes of these modules. The approach will be to build a Backplane and card cage capable of holding four or five RF Modules, i.e., it would have four or five "slots". Each of these slots would be identical and would support the multipin digital connector as well as the RF connectors. This Backplane may be a combination of commercial standard buses such as VME or VXI with a custom section for the LISTS specific digital, IF, & RF signals or a completely custom bus structure. There are at least two approaches which could be used to handle the variable size RF Modules.

One approach would be to confine the RF Module size to an integral number of slots. With this approach, a RF Module that needed two slots would use the connectors on one of the slots for inputs and outputs and leave the other slot connectors unused. An oversize RF Module would take up two slots and reduce the number of RF Modules that could be installed in that Chassis.

A second approach would be to allocate a fixed space to each slot which provided ample room for even very large RF Modules. With this approach, all sizes of RF Modules would have the same size face plate so as to completely close off the front of the card cage.

With either of these approaches, if the embedded CPU is a commercial board such as VME or VXI, then a RF Module "slot" would actually be made up of two sections, one section for the commercial board and a second section for the custom connectors. In such a design, the commercial CPU board would communicate with the rest of the RF Module via multiple parallel and/or serial links carried by spare (undefined) pins on the commercial standard connectors. Figure 3 shows some possible Backplane connector configurations. The first is an example of a full custom design. The second would use standard VME format for P1 and P2 connectors with a custom P3 backplane for the RF signals. The third shows this same approach with the addition of another set of P1/P2 connectors to accommodate a COTS VME processor board for the RF Module embedded CPU.

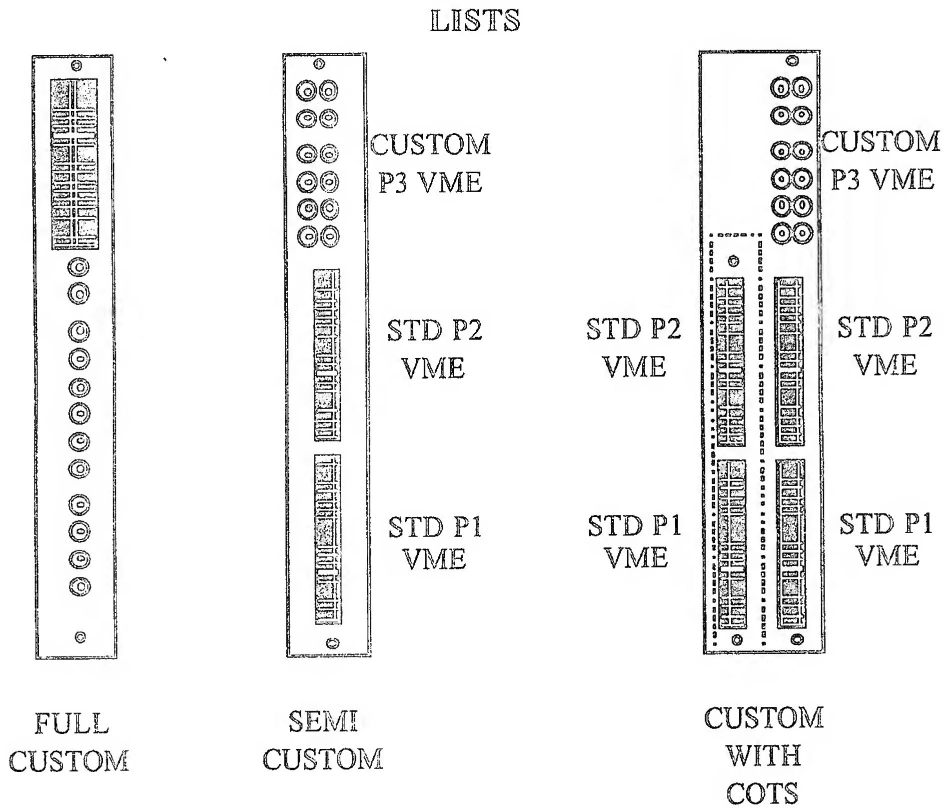


Figure 3: Backplane connector configurations

The decision as to which of the above approaches (or some other approach) to use for the LISTS prototype will be one of the design trade-off studies to be performed.

The LISTS Chassis must also contain the RF circuitry to combine the outputs from the RF Modules into a single set of outputs for presentation to the system under test. Once again, the global requirements mentioned earlier will drive the design. If the global requirements dictate that LISTS must accommodate up to eight RF outputs then the design of the prototype Chassis RF combiner circuit must be able to contain eight channels of combiners, (also, the Backplane must be able to support eight channels of RF connectors to the RF Module). The bandwidth of this RF circuitry must be as wide as possible with a minimum of 2 - 18 GHz. All signal paths for a particular slot must be the same length to preserve phase integrity over frequency.

Controller Software

The Controller CSCI will be the primary user's interface to the LISTS. In addition to the user's interface, the CSCI will support preparation of a simulation exercise, manage exercise data, manage RF Module application code and data, control the loading of application code and data to the RF Module, interactively control the real-time operation of the RF Module and interface with other systems in the reprogramming test environment via a Local Area Network (LAN).

The learning curve associated with a new tool or system can add a significant cost to integrating the system into an existing test environment. To minimize this it would be ideal to design the user interface

to the LISTS to emulate the user's interface of an existing systems simulation set-up. In this context, "emulate the user's interface" implies that the data input to the LISTS Controller would be completely compatible with the existing system and the existing system could even be used to create the simulation exercise. The problem with this approach is that the LISTS will be specifically designed to emulate threats that the current simulation systems can not handle. Thus LISTS will certainly have command parameters which do not exist on the existing system. The emulation of an existing user's interface, while it has appeal from the training aspect, would be unwieldy at best and a disaster at worst. A Graphical Users Interface (GUI) builder will probably be used to develop a new user's interface based loosely on an existing interface. A well designed GUI will have a minimal learning curve.

The preparation of a simulation exercise includes (1) emitter selection/definition, (2) threat site and unit under test (UUT) location and (3) emitter and UUT motion definition. The Controller CSCI will support the ability to create and edit emitter parameters, access pre-existing emitters in the LISTS database, create scripts that define the emitter control during real-time operation and check all entries for legal ranges. The ability to create threat site/emitter and UUT locations and trajectory parameters will be supported by user input or by extraction from the library database. The ability to archive simulation exercises or portions of simulation exercises will also be supported.

The Controller CSCI will support the selection and downloading of the code and data to the RF Modules' embedded CPU. This includes Self-Test, Calibration, and Emulation application programs. The Emulation application will consist of both real-time control code and data. The ability to easily change the data associated with the Emulation application is crucial to the usability of LISTS. There may be more then one Emulation program, so additional cross checks to verify that the Emulation code is consistent with the defined emitter(s) may be required. During the actual real-time execution of the simulation, some level of real-time interaction between the Controller and the RF Module will be required.

The interfacing of the LISTS to other systems in the simulation test environment is a necessity. This includes interfacing to the ADAMS and CATDARS systems and may require interfacing to existing simulation systems such as the Advanced Standard Threat Generator (ASTG) if LISTS will be used to augment an existing scenario by adding advanced threats in a coordinated fashion.

While all of the above functionality is required, the prototype software may be less than comprehensively implemented by the end of the current LISTS program. The intent is to create a robust program structure that can be easily extended. To this end, the programs will be modularly designed to support later additions and new features. The program will be implemented in a higher order language, probably Ada.

RF Module Software

The RF Module CSCI will execute on the embedded CPU contained in the processor block. The CSCI will consist of a permanent bootstrap loader and a loadable application. The bootstrap loader program will be contained in Read Only Memory (ROM) and is the only code and data permanently associated with an RF Module. The bootstrap program will be unclassified. The application will consist of code and data that is loaded to the RF Module via the Controller. The application may be classified, but it will never permanently resident on the RF Module. The application will be lost as soon as power is removed from the RF Module.

The bootstrap loader will be responsible for performing a power-on test of the embedded processor and memory. At the completion of the power-on test, the loader will enter a loop waiting for an application to be sent by the Controller. The loader will be commanded by the Controller to load a program of a given size, at a given memory location and then to transfer to a given memory location at the completion of loading the application. The boot loader will have no intrinsic knowledge of what the application is intended to accomplish.

The application code that will be loaded to the RF Module via the Controller can be of several different types. The Self-Test application will perform diagnostics on the RF Module. The Calibration application will perform measurements to compute the corrections necessary to ensure that the actual RF output power matches the commanded power. The Emulation application will be the actual real-time control code and data that enables the RF Module to emulates a given threat or threats. The data portion of the Emulation application will specify given threat characteristics and will be easily changed. It is uncertain, at this time, that there will be a single Emulation application real-time control code. Different classes of threats may require different real-time control codes. It even may be better to have several different real-time control codes than to have one monolithic program from both an execution speed and the ease of software maintenance aspect.

The Emulation application code will be interactive with the Controller CSCI in order to support real-time simulation exercises. Two types of control are envisioned. The simpler form merely allows the user to start the simulation at a given point in time in concert with starting other simulation equipment to achieve a given simulation exercise. The more complex form allows the controller to coordinate several RF Modules to allow emulation of complex threat sites with several emitters.

In the event that a highly optimizing compiler for a higher order language is not available for the embedded CPU, this software may be developed in assembly language. Ideally, a cross assembler hosted on the Controller will be available.

INTEGRATION WITH ADAMS AND CATDARS

ADAMS Interface

ADAMS will employ the LISTS as a resource in its coordinated testing of an EC system. It will be able to instruct the LISTS to simulate specific emitters with specific characteristics and will receive verification back from the LISTS that these tasks are being performed. The interface between ADAMS and LISTS will be at a minimum some sort of standard protocol bus such as Ethernet, IEEE-488, or 1553B. It could also include discrete digital and/or analog signals necessary for time synchronization.

CATDARS Interface

The CATDARS system will provide a tool to assist the user in generating emitter files compatible with the existing ASTG threat simulator. The LISTS Controller software will also accommodate this emitter file format so that the same files built for the ASTG can be used with the LISTS.

FUTURE EXPANSIONS TO THE LISTS ARCHITECTURE

As a prototyping effort, LISTS is intended to show proof of concept. While every reasonable step will be taken to ensure easy transition to a production version, the programmatic restrictions of schedule and cost will shape the program.

Chassis

The prototype LISTS is defined to work in conjunction with a Radar Warning Receiver (RWR) mechanized with four quadrant antennas and one omni using signal amplitude angle of arrival (AOA) determination. While this is not an uncommon antenna/RWR configuration, it does not cover all of the potential systems. Extending the LISTS to a more robust testing environment will undoubtedly involve modifications to the prototype Chassis design to support alternate antenna configurations. Since each RF Module is a stand alone emitter simulator, it will be possible to design multiple Chassis configurations ranging from a single slot portable chassis to a multi-slot "gang-able" chassis that would allow larger numbers of RF Modules to work in concert with each other.

RF Modules

There will be two prototype LISTS RF Modules built to meet the 4 quadrant/omni antenna configuration with amplitude AOA determination. These RF Modules will support CW and pulsed waveforms with pulse repetition frequencies (PRF's) ranging from 50 Hz to 600 Khz, and binary phase coding. The RF Module architecture will be such that new plug compatible units can be developed at minimized cost. Additional RF Modules covering phase AOA, extended RF frequency ranges, and/or additional output channels can be developed within the existing LISTS architecture.

Controller

The Controller is probably the one part of the LISTS system that is not expected to undergo fundamental changes in transitioning to a production system. While memory and mass storage sizes may be increased, the basic Controller configuration is expected to stay stable. Different packaging styles may be used to support field portable versions versus fixed lab installations.

Software

The prototype LISTS software will be designed to support a demonstration of the RF Module and is not intended to be full production version. The software design will be sufficiently modularized to serve as the basis for a production version. Fleshing out the software to the full functionality of a production system will be required.

The LISTS program is funded by the Wright Laboratory in cooperation with Warner-Robins Air Logistic Command. For additional information on LISTS, please contact Mr. Joe Cardenia at WL/AAAF, WPAFB, OH.

EMBEDDED SOFTWARE SUB-WORKING GROUP (ESSWG)

Co-Chairs:
Chuck Satterthwaite and Steve Hosner

REUSE-BASED SOFTWARE DEVELOPMENT

Dr. Steven R. Christensen

Lockheed Fort Worth Company
Avionic Systems Department MZ-1716
P.O. Box 748 Fort Worth, Texas 76101

I. Introduction

The Department of Defense expects to save billions of dollars annually from improved software technology in three areas. First is working faster with computer-based development tools. Second is by working smarter through software development process improvements. And third is work avoidance through reuse of existing software components. Of these, the largest savings are expected to originate from software reuse. Figure 1 illustrates the additive savings expected for these three software technologies. By the year 2000, software reuse is expected to save $7 billion annually, an amount equal to the other two technologies combined.

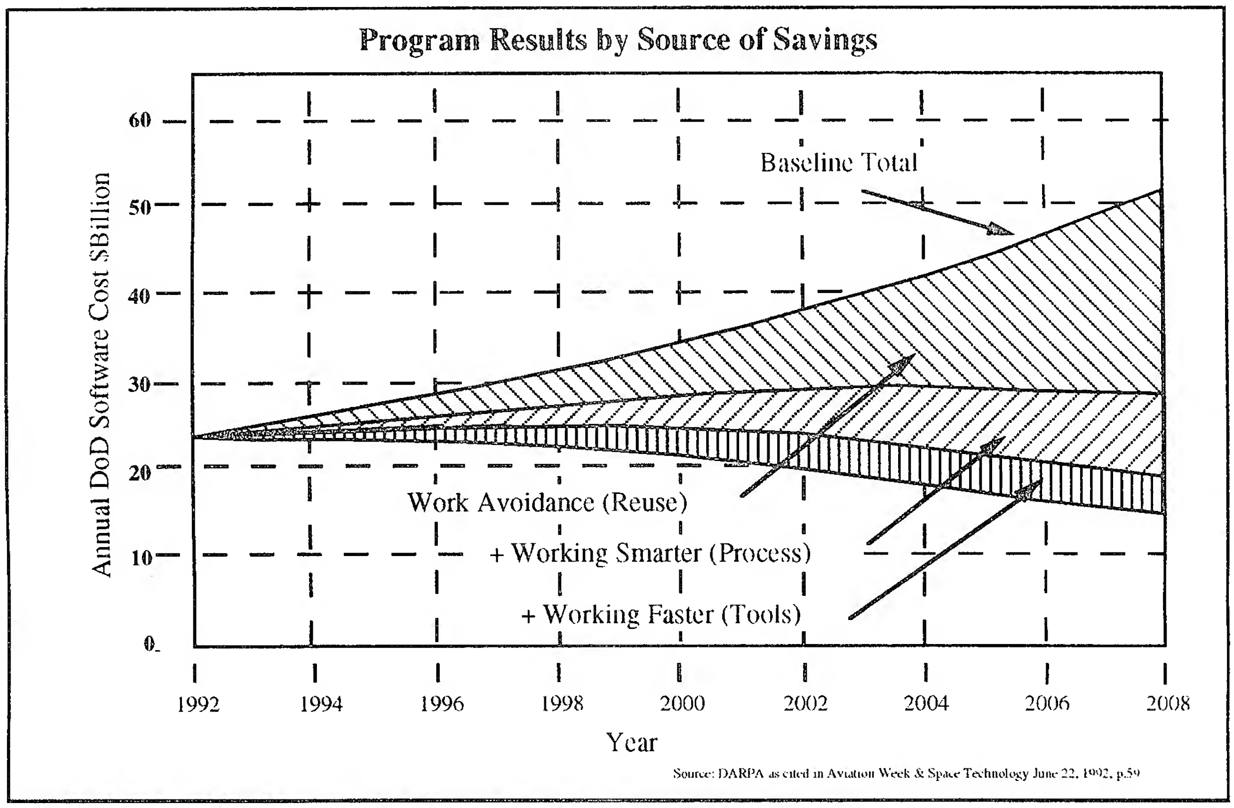


Figure 1. Anticipated Annual Savings from Software Technology

The benefits of software reuse appear to be obvious, almost intuitive, and certainly substantial. For a particular project, the dollar savings can easily be calculated by estimating the cost difference between development without reuse and development reusing existing software components. While this figure may not be precise, it is bound to be reasonably accurate or at least provide an order of magnitude of the savings. A high level of confidence in the software is achieved through familiarity with it and the quality of the software is generally higher than that of a newly developed software component.

The costs associated with software reuse are less obvious and deterministic, but also potentially substantial. Identification of any increase in the cost of designing and coding for reuse is difficult if not impossible. Most techniques for increasing reusability, such as modularity and use of standards, are good software engineering techniques and should be used regardless of the reusability issue. The often stated assumption that designing and programming for reuse is very expensive is more probably an admission that their software development process is immature. This is not to say that there are no costs. The effort required to implement a cataloging scheme for retrieval of software from reuse libraries is frequently not considered. Domain analysis is a foreign concept and requires costly training and manpower. Ownership becomes a new problem requiring legal costs. And software reuse is not without its own set of problems. The integration of software reuse with rapid prototyping introduces new challenges. Software reuse complicates software configuration management. The potential for overestimating cost savings and underestimating cost increases is great as is the reverse. underestimating cost savings and overestimating increases in cost.

The largest savings in reuse are through planned reuse and not opportunistic reuse. The savings are to be achieved through development of a family of products that are designed and built to be reused. This is called leveraged reuse because after initial development of the software components. each subsequent application can leverage off existing software components. This is far different from the opportunistic reuse most often associate with software reuse where software components are reused only when someone is aware of their existence and the opportunity arises to use the component.

The concept of a family of products that share a common set of requirements is the basis for reuse-based software development. This concept is being applied in other applications. For example, the Joint Advanced Strike Technology program (JAST) will develop a new concept of modularity for Navy and Air Force aircraft based on harmonization of requirements and use of a common baseline. The Director of Air Warfare for the Navy, RADM Bennitt, states "I believe that the JAST concept will

lead to a family of aircraft with as much commonality as possible given the rather divergent needs of the Air Force (low-end multi-role fighter) and Navy (high-end, stealthy strike-fighter)." (Bennitt 1993).

The challenge for post-cold war defense contractors is to remain competitive in a shrinking market. The ability to produce low cost yet high quality products is not only a distinct competitive advantage, it is a necessary capability for corporate survival. For software intensive systems that have a common set of requirements from different customers, reuse-based software development has the potential to provide the critical ability to produce low cost and high quality products. However, reuse-based software development is significantly different from traditional software development and requires a significant paradigm shift for both managers and technicians. The issues go beyond technical feasibility to include organizational culture and customer understanding of this new process.

The difference is similar to the difference between manufacturing and assembling. In today's software development environment, software is manufactured. Each new project starts with a set of requirements from which a new design is developed. Under a reuse-based software development approach, each new project starts with a set of requirements that already have solutions in existing components for the majority of the requirements. Some new components may have to be developed and some existing components modified. However, the components, old, new, and modified, are used to assemble the new product. The assembling of software products requires that the component parts be available and designed for easy assembly. Within a well defined domain, a domain analysis will indicate what requirements are common with all products and what requirements are variable for each product. An architecture that supports reuse can be developed and components produced. This effort, termed domain engineering, requires a defined process.

II. Historical Foundations for Reuse-Based Software Development

While the concept of reuse is widespread in software development, a systematic process for reuse has not been widely accepted. As Prieto-Diaz (1993) stated, "The problem we face in software engineering is not a lack of reuse, but a lack of widespread, systematic reuse. Programmers have been reusing code, subroutines, and algorithms since programming was invented. They also know how to adapt and reverse-engineer systems. But they do all this informally."

The basic concepts of Reuse-Based Software Development began to appear with regularity in the late 1980s. An early proponent, Edward Comer stated that domain analysis was the foundation for a reuse-based software system development process and suggested a three phased

approach (Comer 1990). The first phase was to model the domain to obtain greater understanding of the domain of interest. The second phase was to develop an architecture for the products in the domain. The third phase was to develop reusable software components based on the architecture. The end goal of domain analysis was to build and maintain a reuse library of software artifacts which could be used to develop future systems in the domain. Comer (1990) also equated domain analysis to systems engineering of a family of systems in an application domain.

The problem was how to reorganize the software production process to take advantage of work done in previous developments. The proposed solution is viewing system development as creating members of a family of products rather than creating a new product each time requirements change. Once the basis for creating a family of products is established, engineers can concentrate on determining the requirements for a particular product and how the requirements vary from those used to develop the family product. The domain model aids in this process. The new product is developed using the existing domain architecture, saving significant effort in the design phase. Since a large portion of the new system will consist of already developed software artifacts, the development effort is greatly reduce as is the testing effort. The new products are placed in the reuse library for future use.

These basic concepts of engineering a system through a process of domain modeling, domain specific architecture, and developing reusable software components is the conceptual foundation of reuse-based software development in general. The Software Productivity Consortium (SPC) has developed an excellent overview of this process and is presented in SPC's document "Introduction to Synthesis" (SPC 1990).

III. Overview of Reuse-Based Software Development

The basic assumption in reuse-based software development is that for a particular domain the systems within the domain share a large number of common requirements, but that they also vary in well defined ways. When this assumption is not true, reuse-based software development may not be appropriate. When the assumption is true, reuse-based software development provides a methodology for constructing software systems as instances of a family of systems having common requirements. Reuse-based software development's distinguishing features are (SPC 1990):

- Formalization of domains as families of systems that share common requirements but also vary in well-defined ways.
- System building reduced to resolution of requirements, and engineering decisions representing the variations characteristic of a domain.

- Reuse of software artifacts through automated adaptation of system components to implement engineering decisions.

- Model-based analyses of described systems to help understand the implications of system-building decisions and evaluate alternatives.

A family of systems in this context is a set of similar requirements that are satisfied by a common software architecture, and a set of closely related design choices. By focusing the requirements analysis on identifying relations and variations among a set of potential applications. the design architecture can be developed and standardized so that it is easily adaptable to those variations. The engineering of the application consists entirely of refining and evaluating engineering decisions and the automated generation of software artifacts. The process is applicable both to the initial generation of a system and to subsequent variations throughout its existence.

The development of applications under this process is termed *application engineering.* The initial analysis of the requirements, modeling of the domain, development of the architecture, and development of the automated generation system is conducted under a process termed *domain engineering.* These two processes, application engineering and domain engineering, together constitute a new paradigm in software development. that of a two life cycle model. Figure 2 illustrates the interaction between the two life cycles.

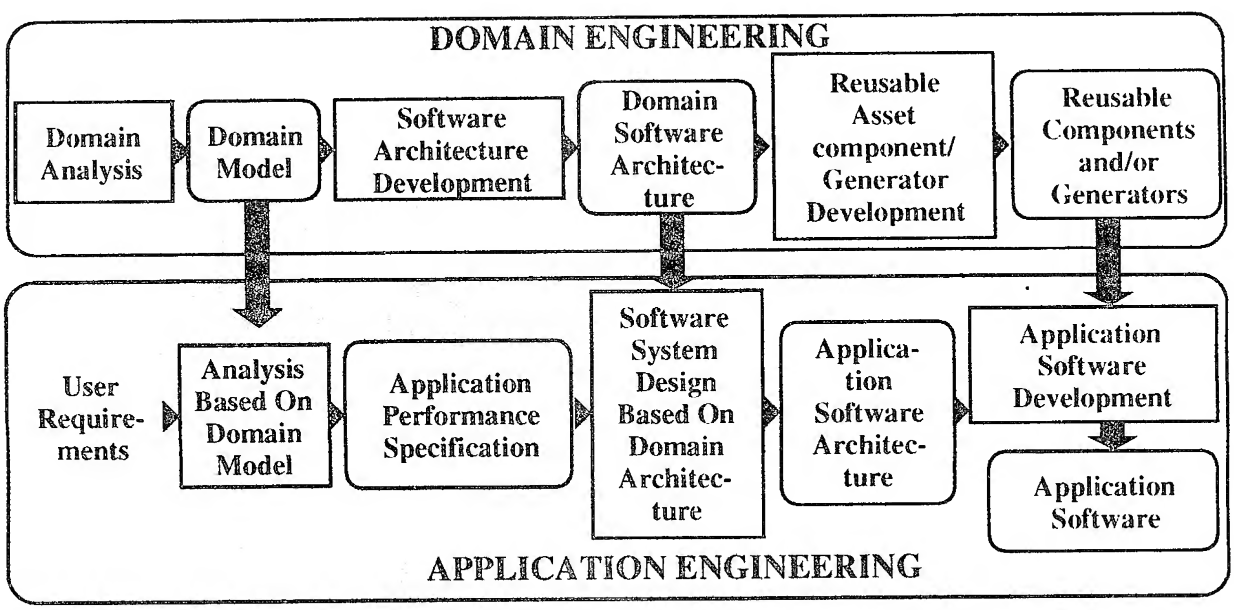


Figure 2. Two Life Cycles of Reuse-Based Software Development

A. Advantages.

Reuse-based software development has several advantages over the more traditional development methodology:

- Capture and documentation of expertise - domain engineering facilitates the capture of domain expertise and records that knowledge in the domain model, architecture, and reusable software components.

- Productivity and quality enhancement - the results of domain engineering are shared among projects and across system development iterations reducing application development effort and errors.

- Ease of change - since the domain model is an explicit form of known domain requirements, only changed or new requirements need be analyzed.

- Manageability - ability to quickly generate a prototype system provides for better risk management, requirements understanding are facilitated by domain model, and domain architecture limits variance in engineering.

B. Principles of reuse-based software development.

The conceptual foundation of reuse-based software development is based on four principles (SPC 1993):

- Product families - a set of products that are sufficiently similar that it is worthwhile to understand the common properties of the set before considering the special properties of individual instances. The creation of new product versions is viewed not as a successive modification to a previous version, but as a derivative from a common abstraction. Each product of a family can be characterized entirely in terms of how it differs from the common abstraction.

- Iterative process - a process in which work products are considered complete only after repetition of the producing and using activities. Since many errors in software work products are difficult to discover without user feedback, an iterative process systematically produces better quality results than a process that depends on producing correct work products without such repetition.

- Specifications - a complete, precise description of the verifiable properties required of a product. A specification is either a description of the requirements (the problem to be solved) or a description of a design (the form and/or content of a solution). The domain model and domain architecture capture the specifications and address the variations that characterize a family of products.

- **Abstraction-based reuse** - provides a means for representing a product family as a set of adaptable work products and for deriving instances of each work product to produce a particular system product.

C. Context for Applying Reuse-based Software Development

Applying reuse-based software development is accomplished in the context of having business objectives and system and software practices which support domain engineering and application engineering.

- **Company business objectives** - determine the types of products built and the customers. The environment most conducive to the effective implementation of reuse-based software development is one where emphasis is given to long-term business objectives. This implies a positive climate for investing in the future, spending money today which will be rewarded with greater savings in the future. It is vital to consider these larger business concerns when addressing the needs of a particular customer to avoid arbitrarily sacrificing longer-term interests to short-term pressures. The long term business objectives can be achieved through an incremental approach where each iteration of a family of products moves the family closer to the long term objectives.

- **System engineering practices** - typically partition the problem and solutions into subsystems. Such a partitioning is compatible with reuse-based software development when a systematic approach is used that results in similar partitioning of similar systems. Reuse-based software development is as much a systems engineering process as it is a software engineering process.

- **Software engineering practices** - typically proceed through analysis, design, development, and evaluation phases. Reuse-based software development modifies this practice by injecting the company business objectives into the requirements phase and by eliminating redundant efforts through systematic reuse of software products. Productivity is increased by focusing on the aspects of the system that are different from the family of products derived from the business objectives.

D. Leveraged Synthesis Process

"A leveraged Synthesis process is one in which strategic business needs. tempered by past project experience and current project needs, define a domain that motivates reuse as a vehicle for achieving long-term organizational objectives. A separate application engineering project is instituted to serve each customer having a problem judged to be within the domain." (SPC 1993, p. Syn-21) Figure 3 illustrates the relationship between domain engineering and application engineering processes.

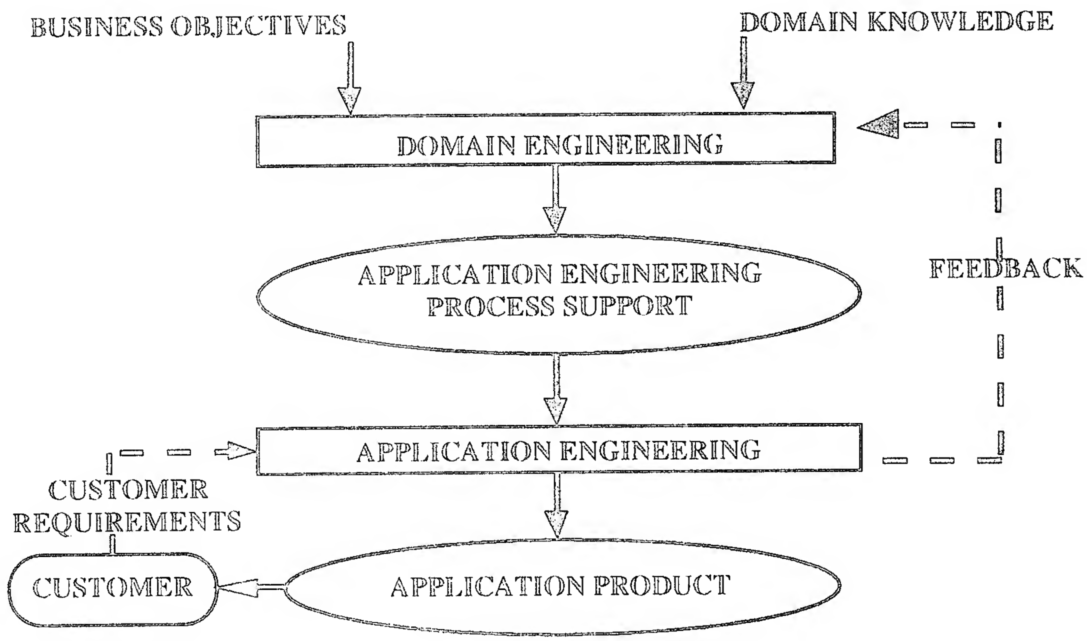


Figure 3. Reuse-Based Software Development Process

E. **Domain engineering** is a series of integrated activities that creates and supports a domain specific model, architecture, reusable components, and an application engineering process support environment. The activities are domain management, domain analysis, domain implementation, and application project support. Figure 4 illustrates the relationship of the domain engineering activities and the four domain engineering work products they produce. The four work products are:

- **Domain Plan** - describes how the domain is expected to evolve through incremental development and defines the individual tasks and resources allocated for each increment.

- **Domain Definition** - defines the scope in terms of classes of systems, characteristics, or functions included and excluded from the domain and how included systems are distinguished from one another.

- **Domain Specification** - formalizes expert knowledge of how to express problems in the domain and how to structure and create corresponding solutions for the problems. A domain specification is a precise characterization of the product family and an application engineering process for constructing members of that family.

- **Domain (Component) Implementation** - is an implementation,

including documentation and automated support, of the Application Engineering process and product family for the domain, as prescribed by the Domain Specification.

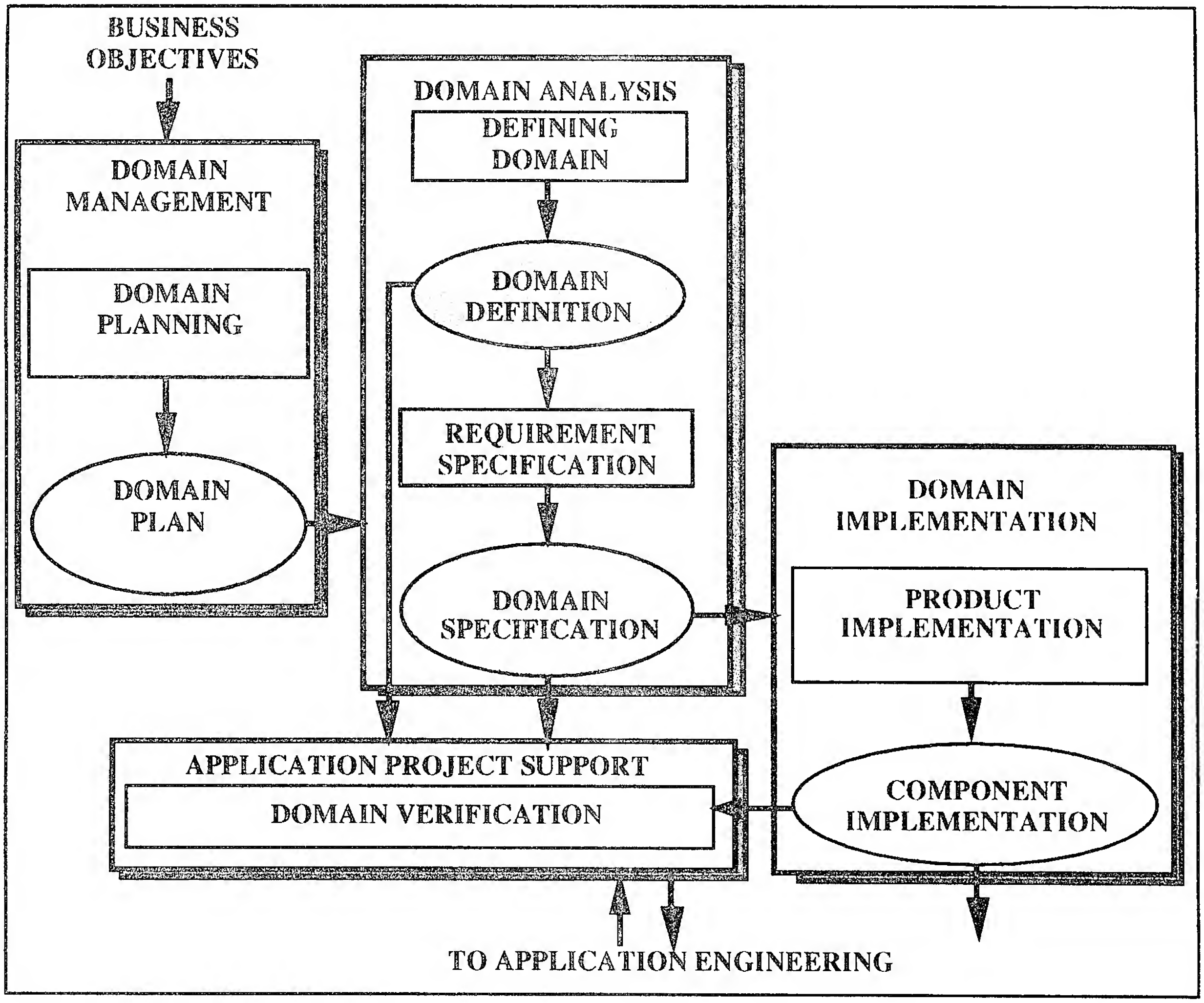


Figure 4. Domain Engineering Products

F. Domain Engineering Activities.

Each of the domain engineering activities are subdivided into sub-tasks, some of which are further divided into lower level tasks. This hierarchical decomposition is illustrated in Figure 5. A detailed description of each task and sub-task is beyond the scope of this discussion. The activities described in the Reuse-Driven Software Process Guidebook (SPC 1993) are not of sufficient detail to conduct a domain engineering activity. At LFWC the activities were modified slightly and given more detail, particularly in the area of format of work products.

<table>
<tr><td colspan="4">DOMAIN MANAGEMENT</td></tr>
<tr><td rowspan="7">DOMAIN ANALYSIS</td><td colspan="3">DOMAIN DEFINITION</td></tr>
<tr><td rowspan="6">DOMAIN SPECIFICATION</td><td colspan="2">DECISION MODEL</td></tr>
<tr><td colspan="2">PROCESS REQUIREMENTS</td></tr>
<tr><td colspan="2">PRODUCT REQUIREMENTS</td></tr>
<tr><td rowspan="3">PRODUCT DESIGN</td><td>PRODUCT ARCHITECTURE</td></tr>
<tr><td>COMPONENT DESIGN</td></tr>
<tr><td>GENERATION DESIGN</td></tr>
<tr><td rowspan="4">DOMAIN IMPLE-MENTATION</td><td rowspan="3">PRODUCT IMPLEMENTATION</td><td colspan="2">COMPONENT IMPLEMENTATION</td></tr>
<tr><td colspan="2">GENERATION IMPLEMENTATION</td></tr>
<tr><td colspan="2">DOMAIN VERIFICATION</td></tr>
<tr><td colspan="3">PROCESS SUPPORT DEVELOPMENT</td></tr>
<tr><td rowspan="2">APPLICATION PROJECT SUPPORT</td><td colspan="3">DOMAIN VALIDATION</td></tr>
<tr><td colspan="3">DOMAIN DELIVERY</td></tr>
</table>

Figure 5. Domain Engineering Process Activities

IV. Lockheed Fort Worth Company's Implementation

The LFWC Software Engineering Process Group (SEPG) Software Reuse Working Group developed a software reuse practice and incorporated it into the standard software development process. However, this reuse practice was based on opportunistic reuse and did not specify the details of a reuse-based software development process (leveraged reuse) and work products. It was decided to develop a domain engineering process under Internal Research and Development (IR&D) funding and conduct a domain engineering effort in the Software Test Station domain. Feedback from the pilot project would allow refinement of the domain engineering process and collection of metrics on the amount of effort required for each domain engineering activity.

The Reuse-Based Software Development research is part of a larger research program in Software Technology. The objective of the Software Technology research program is to improve the capability to develop high quality, complex, large scale embedded software systems consistently and cost-effectively in accordance with contractual commitments. The objective of the Reuse-Based Software Development research effort was to develop and test through a pilot project a reuse-based software

development process that incorporated the dual concepts of domain engineering and application engineering.

The specific problem facing Lockheed Fort Worth Company (LFWC) was the lack of a defined and implementable reuse-based software development process. The existing standard software development process could easily be adapted to the concept of the Application Engineering part of reuse-based software development, but there was not an available domain engineering process.

Early research in existing defined domain engineering processes resulted in the conclusion that the Synthesis process defined by the Software Productivity Consortium (SPC) was the most complete process and was a good framework from which a more detailed and implementable process could be developed. It was also compatible with the efforts of the Department of Defense in Megaprogramming. Therefore the SPC Reuse-Driven Software Processes Guidebook (SPC-92019-CMC November 1993) was used as the process framework and a more detailed addendum was developed for LFWC implementation. Subsequent use of the Synthesis has confirmed that this was a good decision.

Early in the effort it was apparent that those conducting domain engineering needed training in the concepts prior to actually conducting the work. It was not something that was easily picked up by doing the work. The context for a particular domain engineering effort needed to be understood. Once that was done, the domain technicians had no problem in proceeding with the task at hand.

There is a reluctance of projects to look at the larger picture and potential benefits of reuse-based software development. It is analogous to fiefdoms fighting against a united kingdom. This view is somewhat understandable because their careers are at risk if the project fails. Implementation of reuse-based software development will require mandates by top management for project management to take a larger view.

The effort is a multi-year program of education of the software engineers and their managers while the technical issues are being resolved. Reuse researchers at LFWC have already concluded that while the technical and organizational issues will be difficult to resolve, the potential savings in future software development are large.

References

Bennitt, B., "HOWGOZIT," Wings of Gold, Winter 1993, p. 13.

Comer. E., "Domain Analysis: A Systems Approach to Software Reuse," Proceedings of 9th Digital Avionics Systems Conference, 1990.

Nordwall, B.D., "Defense Dept. Expects New Strategy For Improving Software to Save Billions," Aviation Week & Space Technology, 22 June 1992, pp. 59-60.

Prieto-Diaz. R., "Status Report: Software Reusability," IEEE Software, May 1993, p. 61.

SPC-90019-N, "Introduction to Synthesis," Version 01.00.01, June 1990. Software Productivity Consortium, Herndon, Virginia

SPC-92019-CMC, "Reuse-Driven Software Processes Guidebook," Version 02.00.03, November 1993. Software Productivity Consortium, Herndon, Virginia

Hypermedia for Avionics System Software

Brad DiDio
Michael Hutchins
Felsa Satlow
Charles Stark Draper Laboratory
Cambridge, Massachusetts 02139

Marc Pitarys
United States Air Force
Wright Laboratory (WL/AAAF)
WPAFB, OH 45433

Abstract

Under the Hypermedia for Avionics Software Support(HASS) contract with the United States Air Force's Wright Laboratory, the Charles Stark Draper Laboratory developed a prototype that demonstrates how access to technical reference manuals can be improved with the use of a computer utilizing the modern techniques of hypermedia. Such systems enable a user to access materials in various formats on one display station, rather than having to consult a variety of different sources, possibly in different media. Users can navigate easily to review previously seen material, to advance to later steps, or view supporting material. Information is divided among a number of windows so that different types of information is available simultaneously.

The HASS Reference System is a unique information access and presentation tool that electronically converts the *F-16 Avionics System Manual (ASM)* and related material into an interactive "reference base." HASS organizes and presents *information* in a completely interactive manner. What the HASS system offers the user is *control*: You see what you want, when you want it.

The system is based on a unique information access and presentation framework that electronically converts a set of related documents and other material (e.g. video demonstrations, technical drawings, computer animation) into an interactive "reference base", and provides separation between the *information* and the *presentation.*

The HASS system was developed on a Macintosh platform. It uses SuperCard and an Oracle database. It is designed to support software engineers testing modifications made to operational fligth software. This software testing is performed on a Sun computer-based system called the Advanced Multi-Purpose Support Environment (AMPSE). Access to the HASS system can either be from the Macintosh on which it runs, from the AMPSE, or from another Macintosh, possibly located remotely. This means that the information is not only available to the software testers, but also to other diverse users, possibly located at other sites. It is envisioned that the system could be expanded by creating other databases, possibly at other locations, which could be accessed by the same users, using the same software. These other databases would be accessible over a TCP/IP Network protocol which could allow access over the Internet. Each database would act as a database server, with clients running the HASS software. Information would be distributed and accessed in a manner similar to the World Wide Web and Mosaic. The main distinctions are in the forms of access provided, including both full text search and conceptual indexing, and in the lack of *hot spots*, relying instead on *dynamic linking*.

The HASS Framework

The HASS system is not a database, nor an automated page flipper, nor a set of files that are searched with a word processor. It provides an environment for the hypermedia presentation of information on personal computers. Text, figures, movies, and audio documentation can be accessed quickly and effortlessly.

The system is based on a unique information access and presentation framework that electronically converts a set of related documents and other material into an interactive "reference base", and provides separation between the *information* and the *presentation.*

The HASS framework consists of four parts: the Document Loader, the Reference System, the Database, and the Mac Network Server (MNS). Using the Document Loader, an author loads text, video, and audio, relating to a particular project into the Database. The Reference System retrieves and presents the desired information stored in the Database. The hypermedia links are created dynamically using standard database queries and require no maintenance.

The Mac Network Server is a separate application running on the same machine as the HASS Reference System. It receives commands from other applications, and sends them to the HASS Reference System to be executed. This application starts up automatically when the HASS Reference System is launched.

The information is managed using a Relational Database Management System (RDBMS). This system is made up of a Server and several Clients. The server is the host system that runs the RDBMS and maintains at least one Database that can be shared by remote clients. The client is a system that connects to the shared Database(s) on the server. The client stores and retrieves information using the industry standard language SQL to define and manipulate data. The clients receive packets of data from the database via the Transmission Control Protocol (TCP) and the Internet Protocol (IP).

The HASS framework supports:

- Automatic loading of marked-up documents into a database in a manner that allows retrieval by sections.
- Automatic retrieval of figures and tables by name from folders where they are stored.
- Full-text search made faster by pre-processing the text.
- Conceptual indexing support for contextual access of information by subject.
- QuickTime movies.
- Access of movie clips from laser discs.
- Automatic retrieval of text, figures, tables, or movies by selecting text within the document.
- Easy addition and modification of documents.
- Sharing of information among several users.
- SUN to Macintosh Network link for sending commands to the HASS Reference System from other applications.

The information is stored in a database and is retrieved at presentation. The HASS presentation system offers the reader *control*: You see what you want, when you want it. To understand how this is accomplished, it is necessary to describe two HASS features: "windows" and "menus."

The HASS system presents information in windows. A window is exactly what it sounds like - a way of seeing into the information contained within the reference base. Each logical *type* of information (i.e., text, figure, video, etc.) is given its own window, which is only opened when that type of information is requested. Since a reader will often want to see more than one type of information at a time, several windows can be opened

simultaneously, and positioned wherever desired. When the reader is finished using the information in a window, it can be closed.

The reference base, as it is presented in windows, contains both fixed and "pop-up" *menus*. A menu allows the reader to get more detail on, or material related to, the information that is being read. Menus can lead the reader from window to window. The reader can "select" the text about which more information is desired, and a menu will offer choices of the type of information needed. Menus are used to link related material and to allow the reader to control the type of information for a particular topic, and follow links to related material. The reader, not the authors, decides which path to follow and what to ignore. Textual information may be read linearly (as in a printed book), however, the links to related materials are there when desired.

In effect, the reader is interacting directly with the content material, not with a command line or a word processor. This style of interaction is very easy to learn and use. The reader may need an overview of the information contained within it, or may want to delve into the lowest level of detail without accessing any explanatory or introductory information. There are a number of different ways in which to access the reference base, and they will be described briefly in this paper.

Information Retrieval

The immense appeal of hypertext in general, is due to two major factors:

(1) The ability to jump directly and immediately from one piece of information to another, regardless of the boundaries (e.g., document) that would separate them in classical information-access situations (e.g. from a passage in one document to an illustration in a completely different document)

(2) Having information accessible on any subject that is related to the one which you have accessed.

If the latter factor is analyzed, it can be seen that the kind of relationship that is of real value, especially in the context of technical reference manuals, is the same kind of relationship that is captured in classical indexes by heading-subheading relationships and by "See ..." and "See also ..." references. In other words, what hypertext readers have been enamored of all along is a form of indexing.

During the HASS project, we realized that the classical hypertext form of indexing is not the best form, especially for the high degree of systematic organization that is required to support access to relevant information in technical reference manuals. To see this, consider the classical link more closely: It is typically from one place in the text, where something is referred to but is not the primary topic of the passage, to *the* place where it is discussed substantively. This latter kind of reference resembles that to which a classical index entry points. Integrating the hypertext approach with the indexing approach, something more useful than the classical "point to same kind of point" hypertext link is a link that has the same beginning point, but points to an entry in a back-of-the-book type of index, instead of to just another place in the text. With this kind of link, a reader could jump not to another place, but to a higher-level that shows where other related passages of substantive information can be found. This can be taken one step further: Instead of doing this with links, entities that are present only for certain passages and whose presence needs to be noted explicitly (e.g., by some visual indication), one could imagine just tagging (invisibly, behind the scenes) every passage as to its topic, and thus be able to service any request for more information on any referenced subject. Then, explicit, individual tagging is necessary only for passages whose references could not be determined automatically.

This is the approach we took in developing the Index-Based Information Access (IBIA) part of HASS. One of the most important ways in which we saved large amounts of time for what otherwise would have needed armies of link-creating classical-hypertext indexers is by realizing that, given the controlled terminology to be expected in a good technical reference manual, one ought to be able to determine what concept a term is referring to *automatically*, using a relatively simple, general computational process. In other words, this computational

process can be used instead of making a link from each occurrence of every term whose concept should have an index entry or explicitly tagging each such occurrence as to its concept.

Thus, we have a unification of classical hypertext, classical indexing, and heuristic computational processes: Given today's very limited capabilities for automatically discerning the topic of a passage, most such characterizing was an intellectual process which was done manually. This intellectual subject characterizing has to be represented by some explicit tag, such as a link endpoint or an associational marking. In addition, the notion of a link can be usefully generalized to tagging to indicate subject matter, where the tag is a choice from a controlled, organized, systematically-developed list of topics that refer (implicitly) to an index entry. This approach also adapts quite easily to SGML marked-up text and CALS.

There is an enormous amount of work going on today in the hot areas of Automated Indexing, Free-Text Information Retrieval, and Natural Language Understanding. Hype and promises are flying as fast as they were for Artificial Intelligence in the '70s and '80s -- but the actual performance of existing systems remains far short of the quality of access to information that the user of a Technical Reference Manual ("TRM") requires. Still, the cost of building a conceptual index is so high that investigating what automated indexing of real utility can be done is well-motivated.

We realized that in our context of existing TRMs, the most cost-effective way of automating the indexing process would be to extract from the body of a manual the significant organizing structures that are typically found in TRMs. These structures give fairly good indications of what, and where, are the important concepts and topics covered in the TRM. We focused on this, rather than on techniques for general text.

Automated indexing of a document breaks down naturally into two parts:

(1) Determining what are the important concepts and topics covered in the manual (and what terminology is used to refer to them), and...

(2) Determining where each is substantively discussed (as distinguished from the much larger number of incidental mentions of the concepts).

Determining the Important Concepts and Terminology

The important concepts in a TRM are represented by terminology in ways that allow them to be distinguished from the more common English words of the TRM's text.

Almost all automatic indexing based systems are unable to distinguish words from concepts, yet two such distinctions are crucial for providing quality support for subject access: multi-word terms and synonyms. Taking advantage of the information that's implicit in TRMs, we are able to support both of these automatically.

We identified 5 techniques for determining the important concepts and terminology of a TRM:

(1) Processing the TRM Glossary
Since the purpose of a glossary is to explain (some of) the terms found in a TRM, we can infer that each term explained in a glossary represents a concept that's important enough for us to record in our list of concepts.

(2) Finding all of the acronyms in the body of the manual itself
The nature of TRMs is such that any acronym that's found is very likely to represent a concept that's discussed substantively somewhere in the manual.

(3) Finding all constructs of the form X (Y), where one of the variables is an acronym and the other is the fully elaborated term for that concept
This is a relatively common technique in TRMs for defining acronyms, in addition to assembling such information in a glossary. The first place each acronym occurs, its elaboration is placed after it in parentheses, e.g. "CARA (Combined Altitude Radar Altimeter)". Similarly, frequently the text will tell the reader that, instead of using the fully-elaborated term, its acronym will be

used, e.g. "Combined Altitude Radar Altimeter (CARA)". Both of these lexical forms tell us that this is a concept that is probably important, but also what its acronym and fully-elaborated term is.

(4) Finding all contiguous sequences of initial-cap words
In technical writing, it is common to capitalize terms that stand for concepts that are discussed in the document in question. That means that the appearance of a capitalized word or sequence of words at other than the beginning of a sentence is good evidence that the term represents a concept that's important in the document.

(5) Processing any existing index
Obviously, having a back-of-the-book index would be a gold mine for automatically finding the terms and concepts that are important in a document. However, we did not implement an automated indexing method for this project, because we did not have one.

We implemented the first four of these.

Identifying the Substantive Occurrences of Information

Once the concepts of a TRM terminology have been determined, we can proceed to identifying where in the TRM substantive information occurs for each concept.

For this second aspect of automated indexing, we identified two techniques:

Occurrence of the known terms in the section headers
Concentrations in the distribution of the terms for a concept in the text itself

The first of these techniques is just a straightforward decoding of one of the most basic conventions for TRM organizational layout. Given that section headers are used to announce the main topic(s) of each of the primary divisions of the body text of TRMs, a good heuristic is that the appearance of a known term in a section header is a good indication that the text of that section contains substantive information about that term's concept.

The second technique is based on probably the most well-known "Relevance"-estimating heuristic in Information Retrieval, Gerald Salton's "term frequency times inverse document frequency" ("tf*idf") technique. We implemented a very simple version of this (just considering three or more occurrences of a known term in a section to be sufficient evidence of that section's containing sufficiently substantive information about that term's concept).

Remote Access

The HASS system uses an ORACLE RDBMS which will allow multiple clients, in various locations to access the information in the Database. The Database Server can reside at one location under Configuration Management, while users at other sites can access the information by simply connecting to the database over the Internet (See Figure 1.). While we have not yet implemented it, such a system could provide read/write restrictions to lock out other users while one user is editing a document, so that no one else can retrieve that information. This is critical so that the information retrieved is always the latest version of a document.

Although SQL commands are used to access the database, the user does not need to have any knowledge of the SQL language because the SQL commands are embedded in the functionality of the HASS system. One of the most attractive features of the HASS system is the fact that it is very easy to use.

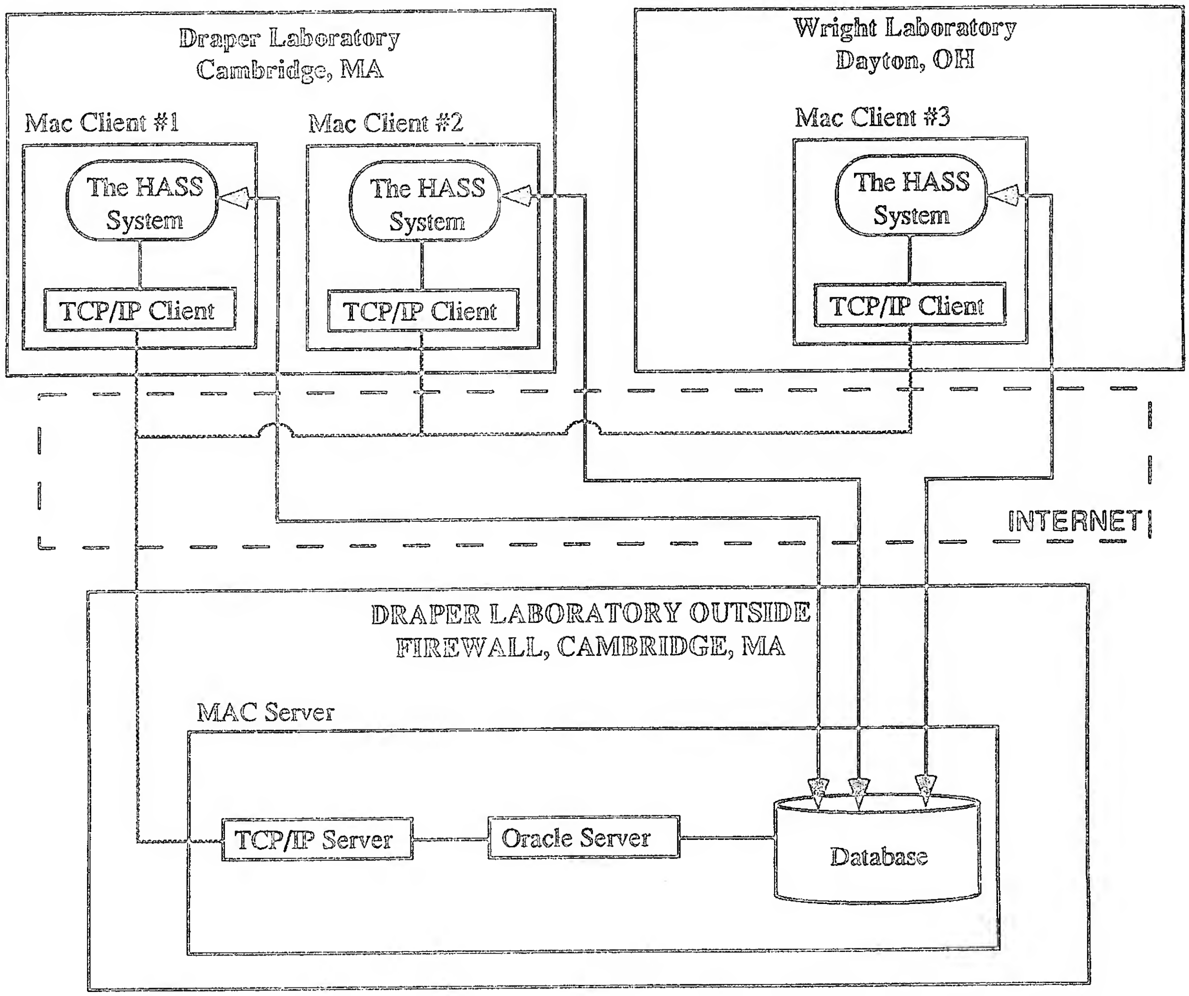


Figure 1. Distributed HASS system with Multiple Clients

HASS Application

The F-16 Avionics System Manual (ASM) was scanned and imported into the framework using the Document Loader. The text was processed through Optical Character Recognition (OCR) software, and the figures were stored in raster format. A few figures were identified as appearing many times, with slight variations. (For example, the Fire Control/Navigation Panel appears over 500 times in the text, with the dials in different positions, requiring different descriptions.) These figures were drawn, so that the dials and switches could be identified as "objects" and manipulated as such. In the resulting figure, the user can point and click at a dial position, the dial will move, and a textual description of the function being performed will appear on the screen.

Using the information retrieval techniques discussed above, we were able to produce quite a respectable index for the ASM, consisting of 605 concepts, 865 terms, 650 indexed passages, and 1285 index entries.

Along with the document access features of the HASS Reference System, it can also receive commands from other Application Program Interfaces (APIs). The HASS Reference System currently only accepts commands from the Sun-based Advanced Multi-Purpose Support Environment (AMPSE) system. The AMPSE has an F-16 Simulator used by software engineers testing modifications for the F-16 fire control computer software. An AMPSE user may request documentation on the current situation of the AMPSE by selecting the Documentation Mode switch on the Simulator screen. The MNS is an application that is run with the HASS Reference System

that continually polls or "listens" for a network connection. The HASS Remote Interface (HRI) is an extension of the AMPSE, which establishes the connection between the AMPSE and the MNS. Once the connection to the MNS has been established, an acknowledgment is sent back to the HRI informing the HRI to begin sending the commands. The HRI then sends a preformatted script file to the MNS which the MNS uses to activate functionality in the HASS Reference System. In this case, the script contains HyperTalk commands that are associated with the HASS Reference System. Parameters are added to the commands dynamically in AMPSE documentation mode. These commands activate a database query to retrieve and present information relating to the current situation of the AMPSE system (See Figure 2.). The script could contain other commands such as Apple events[1] that would work with other applications.

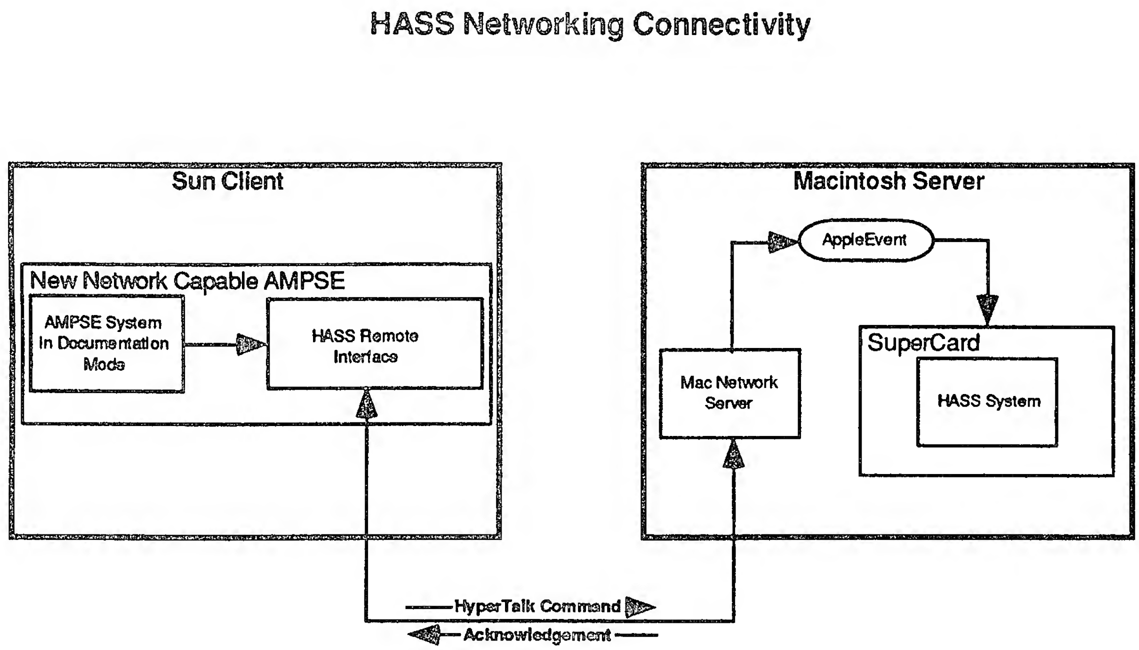


Figure 2. HASS Remote Interface Diagram

The HASS system was developed on a Macintosh platform, and requires two monitors to display all the information that may be requested at one time. A Laser disc player was used for the video components, but capability for QuickTime movies, stored on the hard disk were also provided. Additional equipment is required for authoring, a scanner, since most documentation is currently available only in hard copy, and a PC VCR, to allow the capture of video tape clips into QuickTime movies.

---

[1]Apple events are Apple's protocol governing a class of high-level events.

Conclusion

Draper has combined commercial off-the-shelf software products to produce hypermedia systems. SuperCard™ by Allegiant Technologies, Inc. was the hypertext authoring software used, and Oracle for Macintosh 2.0 was the relational database used.

Documents were scanned with the scanner, and OmniPage Optical Character Recognition software by Caere was used to convert the scanned images into ASCII text. Microsoft Word was used to mark up the text, with the intention of migrating to CALS markup for importing documents with such markup. Figures were scanned and stored in folders, and movies were captured and converted to QuickTime clips using SimplePlayer, an Apple software product.

Included with our software are four reference manuals that can be used to help the user utilize the capabilities of this system. These manuals instruct the user how to install and configure all of the hardware and software components, how to markup and load documents, how to access that information once it has been loaded, and how the application can be modified so they can add functionality themselves if they want.

The goal of both Draper and Wright Laboratory in developing hypermedia systems is to integrate off-the-shelf hardware and a number of software packages together, with Draper expertise. Areas of potential application for this technology include training, reference systems, diagnostics and maintenance, disaster preparedness, or command and control.

Avionics Software Design Complexity Measure[1]

Stephen Graff *SYMVIONICS, Inc.*, Pasadena, California
Marc Pitarys AAAF, Wright Patterson AFB, Ohio

Abstract

The cost to develop and maintain software is increasing at a rapid rate. Of the total cost to develop software, the majority of the cost is spent in the post-deployment/maintenance phase of the software life cycle. In order to reduce development costs, more effort needs to be spent in the early phases of the software life cycle. One characteristic that merits investigation is the complexity of the software design. Since an improvement in the design phase of a software product would reduce the overall cost of that software by identifying problems early, it makes sense to develop a measure which addresses *software design complexity,* which would produce software designs that will be more supportable and maintainable. The ability to objectively evaluate and measure the software design complexity would allow a software developer to evaluate and compare competing candidate designs prior to any coding activities. Comparing designs early in the software design cycle would then permit an early evaluation of the anticipated maintainability, quality and reliability of the final software product. However, software complexity measures are currently focused on the analysis of completed code. This study surveyed design methodologies from traditional and structured analysis methodologies to object-oriented analysis and design. The causes of code and design complexity were investigated with an emphasis on real-time avionics software design issues. The existing measures and metrics were evaluated for use and adaptability into design metrics and on mathematical soundness of the design metric. The study found measures which are useful for measuring from small systems to large systems, and useful for evaluating parts of designs for all domains: avionics, real-time, non-real-time, business, scientific, mainframe, desktop, and embedded.

Life Cycle Issues

The cost to develop and maintain software is increasing at a rapid rate. Of the total cost to develop software, the majority is spent in the post-deployment/maintenance phase of the software life cycle. In order to reduce development costs, more effort needs to be spent in the early phases of the software life cycle. One characteristic that merits investigation is the complexity of the software design. Specifically, the complexity of the software design needs to be measured and minimized via software measures, which would produce software designs that will be more supportable and maintainable. By reducing the design complexity the overall complexity will be reduced. Furthermore, software reliability would be increased, thus reducing maintenance costs. Currently, metric analysis is performed primarily at the end of the implementation level (code) phase of the life cycle, if at all. If a design change is necessary, the entire design and development effort would be repeated, resulting in potentially endless iterations of design/code redevelopment. The objective of the Avionics Software Design Complexity Measure (ASDCM) program is to shorten this iterative process by analyzing the design of the software, thus moving the metrics assessment to earlier in the software life cycle.

This measure will provide software developers with the capability to remove errors earlier in the life cycle at a time when it is less expensive to do so. It will also allow designers and project management objective measures to use for evaluation of design options based on risk and

[1]This study was supported by Air Force Materiel Command (AFMC) WL/AAAF WPAFB through the Small Business Innovation Research (SBIR) Program for Topic AF93-109 via Contract Number F33615-93-C-1257.

reliability assessments. The knowledge of risk and reliability issues provide a clear view of scheduling, manpower, and costs over the life cycle of the project.

Complexities and Measures

Several factors make up design complexity: level of design, cohesion, coupling, and code reuse. To reduce complexity, the cohesion and code reuse should be maximized, and the coupling minimized. These abstractions of complexity apply across all types of software domains including avionics software, real–time software, embedded software, information systems, and operating systems. Each domain has its own type of programming and software engineering problems, however the complexities are language, design methodology and problem independent. Within each domain, both designs and code may be evaluated for complexity and directly compared. Across domains, both designs and code may be evaluated for relative complexity. While those factors that cause real–time complexities are the same factors that cause complexities in other domains, the real–time complexities should be measured separately.

During the design phase, the greater the level of design depth, the better the understanding of the design, the lower the risks, and the later costs due to these risks. Trivial and unneeded design levels weigh down the system, drive up the cost, and provide additional opportunities to have code failure.

Cohesion measures the binding or degree that each module performs on a single, problem related, well–understood function. The highest degree of cohesion is when a module performs one function. This is one of the goals of the structured analysis methodology. According to Coad, one function per module is desirable for object–oriented designs and programs, too.[2] At the highest level of cohesion, the process has been decomposed to the lowest functional level thus achieving design completeness for that function.

McCabe Cyclomatic Complexity is designed to indicate testability and maintainability by calculating the number of threads in existing software and measures the relative complexity of each thread. It is quite useful for isolating and eliminating complex paths, although it does not evaluate the complexity of the parts of algorithms which do not branch. It is used for code which is been developed and not for designs prior to implementation.

Coupling measures the relative independence among multiple modules. Page–Jones gives three reasons why low coupling should be a goal:

1) the fewer the number connections between modules, the less chance of a failure in one module to propagate;
2) the fewer the number connections between modules, the less chance a change in one module will cause problems in another and therefore increasing reusability; and
3) the fewer the number connections between modules, the easier the learning curve is for the programmer to learn about other modules.[3]

Modularization minimizes coupling, thus facilitating the understanding of how modules function and lessen the devastating ripple effect so common in some earlier programs. This ripple effect has a reputation of driving up system maintenance costs.

Kernighan and Plauger in *The Elements of Programming Style* and *Software Tools* discuss code reuse via building tools and building tools based on tools for economy of design effort and ease of understanding code. Obviously, when the subroutines or procedures are built to be more flexible

---

[2]Coad, P., "OOD Criteria, part 1", *Journal of Object-Oriented Programming,* Volume 4, Number 3, (June, 1991), pp. 69-70.

[3]Page-Jones, M., *The Practical Guide to Structured System Design,* (New York: Yourdon Press, Prentice-Hall,1980).

and robust to increase the reuse of the code, the quality of the code improves. The addition of code reuse reduces the initial testing and the cost of fielding the system. An added bonus is the later overall maintenance cost reduction because it is easier for the maintainer of code to understand the code and modify the code. Code reuse simplifies the code and system regression testing.

Survey of Design Methodologies

The design issues for real–time avionics systems are different from other types of systems design because of the presence of concurrency and the timing requirements that exist between inputs and outputs. The system must be decomposed into sets of concurrent processes in a manner that clearly addresses the system complexities and concurrencies as well as the interfaces. The interfaces form the communication and the synchronization of the sensors and the actuators with an emphasis on the timing. Timing is the dominant constraint in an avionics system. Some functions must be carried out periodically while others are interrupt driven. All of which must be accomplished within critical time margins. The system must complete all tasks, validate the data, detect faults, and correct faults. The design must of course demonstrate feasibility.

Structured Analysis

Traditional system design in the 1960's led to a number of large and costly software failures. This was in part due to a lack of consistent system design techniques. During the mid–seventies two design methodologies emerged, one was data structure based and the other was data flow based.[4] The data structure based methodology emphasized refinement and full understanding of the data structures before developing the data processing design structure. Jackson Structured Programming and Warner–Orr methodologies are the two major methodologies of this class.

The data flow based approach was one of the first comprehensive and well documented methodologies to appear. The data flow was developed in a systematic approach and then was mapped into charts. This methodology introduced coupling and cohesion criteria and emphasized functional decomposition into modules with detailed analysis of the interfaces. This process was refined by Tom DeMarco into Structured Analysis. Structured Analysis was developed to bring Software Engineering into the Systems Engineering world. Early structured analysis methodologies include DeMarco, F–Nets, Ganel–Sarson, Martin Action Diagrams, R–Nets, Structured Charts, State Machine Charts, and Yourdon methodologies.

When designing real–time systems with multiple processors, the decomposition may be broken into tasks and subsystems. Tasks run on single processors and subsystems consist of multiple tasks. Process controls are mapped by state transition diagrams by using controls, events and triggers to start processes and transformations. Both related sequential processing and processes, which are triggered by the same events, should also be grouped. Periodic execution based on set timing intervals should be set up as separate tasks. The task separation leads to more comprehensible designs and easier check–out and maintenance. Time critical events must be separated and carefully evaluated for all contingencies, exceptions, and faults, and priorities need to be set.

Object–Oriented Analysis and Design

Object–Oriented Analysis (OOA) and Design (OOAD) was proposed by Grady Booch in 1985 as the outgrowth of the combination of structured analysis and design and real–time controls. Functional analysis allows several modules to operate on an object, while object–oriented analysis allows only one module to have access, control and the power to transform an object. The result is that this methodology coerces high cohesion and low coupling.

---

[4]Gomaa, H., *Software Design Methods for Concurrent and Real-Time Systems*, Reading, MA: Addison-Wesley Publishing Company, 1993) p.37.

There is no agreement on what object oriented analysis, design, programming, and methods are. However, inheritance, polymorphism, and dynamic binding are common threads. In object–oriented programming each class (package in Ada) must have its own objects (for example, data items C++ requires a minimum of two) and at least one process (method), which uses those objects. Classes may have subclasses which inherit data attributes and processes. The objects may have additional attributes assigned at descendant levels. Inherited processed may be redefined, that is, the parent has a process with the same name as the child however each process may be different as long as the data structure can be used as a discriminate between the definitions during compilation or at run time. The compiler assigns the proper process using the data class as the discriminate, if possible, otherwise the assignment is done by run time binding. The multiple definitions for the same process name is called polymorphism and is a kin to overloading in Ada.

If a class that is outside the objects' class, which the outside class needs information about the object or which the outside class needs to process the object, then the outside class calls the proper routine via the owning class (sends a message) with its request.

Object–oriented design uses encapsulation, message sending, polymorphism and inheritance to minimize the coupling and maximize the cohesion of the design.

Beyond object identification there is little agreement on what other functions are involved with object–oriented analysis and design. However there is commonalty in the final structure of the design: objects, structure of classes, description of classes.

Commonalties of Methodologies

Fortunately many roads lead to a common point. Design methodologies provide various analytical strengths, insights and a richness of information. The Structured Analysis, Structured Analysis with Control and Object–Oriented Analysis and Design methodologies usually have these five tools in common under various names: Data Flow Diagrams, Data Dictionary, Process Specifications, Entity–Relationship Diagrams, and State Transition Diagrams.

These methodologies grew from common roots in structured analysis and therefore the ability to tap the databases or repositories of CASE tools for the purpose of deriving the design measures is possible. This repository can be searched for the factors needed for the design complexity measures.

The Design Measure

A measure must show what the complexities are and where they are located. The measure must show an intuitive value for the abstract feature of the design part it is evaluating. In addition, the measure should increase as the complexity increases and the increase should be proportional to the increase in the complexity. As stated earlier, the selected measure must be useful early in the life cycle and continue to be useful throughout the life cycle of the software product, therefore effecting an overall cost savings and increasing software reliability. This measure must: 1) be universally applicable across as many design methodologies as possible; 2) provide an intuitive relative measure of complexity, risk and design completeness; 3) cover multiple aspects to achieve a total picture of complexity, risk and design completeness; and 4) if a complexity measure is high, then the design should be modified to reduce complexity unless the complexity can be justified.

In order to be useful the measure must not view the design from only one point of view or one level; it must be multiview, as discussed by Porter and Selby,[5] and others.

The design measure was developed by combining the factors for the real–time constructs measure, decomposition levels, code reuse, coupling, interface complexity, and cohesion into a

[5] Porter, A. A. and Selby, R. W., *op. cit.*, pp. 46-54.

table. The cyclomatic measure only applies to data flow since the coupling is handles the data and controls flows separately. By presenting the data in a table form, the type of complexity and the complexity for each transform will be displayed. A large values will be easily detected. Summations of these measures can be weighted and the total will provide an overall measure. The results may also be displayed in a multiple metric graph.

$$FOM = aDM\left[-b\sum REU + c\sum CO + d\sum RT + f\sum IC + g\sum COH\right]$$

where:			
	FOM	≡	Figure of Measure
	a	≡	Constant
	b	≡	Constant
	c	≡	Constant
	COH	≡	Cohesion Measure
	CO	≡	Coupling Measure
	d	≡	Constant
	DM	≡	Depth Measure
	f	≡	Constant
	g	≡	Constant
	IC	≡	Internal Complexity
	REU	≡	Number of Routine Reuses
	RT	≡	Real–Time Construct Measure

The Real–Time Construct Measure works exactly like the Coupling Measure. The Coupling Measure is used for data flows and the Real–Time Constructs are used for the control flows. Constants a, b, c, d, f, and g will have to be determined through extensive testing. For this study $a = b = c = d = f = g = 1$. At this point the constants can not be defended and are only used as an illustrative value.

Examples

The first example is taken from "Formalizing Specification Modeling in OOA" by S. Honiden, N. Kotaka, and Y. Kishimoto.[6] The top line of Table 1 gives the titles for the highest level of coupling, number of coupling connections, cohesion measure and Cohesion.

The results are shown in Table 1. The REU equals 25, the coupling measure is 512, real–time coupling, RT, is 0, and interface complexity, IC, is 57. The cohesion measure, COH, is 78.06 which is large due primarily to the lack of cohesion of processes in 1.0, 5.0 and 6.0. This model has three levels of decomposition, therefore the Depth Measure, DM, equals 0.887. The total measure is:

$$FOM = (0.887)(-25 + 512 + 0 + 57 + 78.06) = 551.77$$

The second example is taken from the appendix of *Modern Structured Analysis* by E. Yourdon.[7] This model, which was developed by Dennis Stripe, was used in a workshop sponsored by the Washington, DC, chapter of the ACM in 1986. It is a model of an elevator scheduler and controller. The author wrote: "Its primary purpose is to illustrate the use of structured analysis

---

[6] Honiden, S.; Kotaka, N.; and Kishimoto, Y. "Formalizing Specification Modeling in OOA", *IEEE Software*, January, 1993), pp. 54-66.

[7] Yourdon, E., *Modern Structured Analysis*, (Englewood Cliffs, New Jersey: Yourdon Press, Prentice-Hall, 1989), pp. 631-659.

models for real–time systems; you will see examples of control flows, control processes, and state–transition diagrams that would typically *not* be used in a business–oriented system."

Process Number	Coupling Measure	Cohesion Measure
Totals 1.0		24.27
1.1	8	4.76
1.2	54	15.51
1.3	45	2
1.4	3	2
Totals 2.0		4.76
2.1	36	4.76
2.2	27	0
Totals 3.0		6.76
3.1	27	0
3.2	36	0
3.3	27	0
3.4	27	4.76
3.5	27	2
Totals 4.0		0
4.1		0
4.1.1	18	0
4.1.2	9	0
4.1.3	4	0
4.1.4	4	0
4.2		0
4.2.1	9	0
4.2.2	4	0
4.2.3	4	0
4.2.4	2	0
4.3		0
4.3.1	8	0
4.3.2	4	0
Totals 5.0		25.51
5.1	18	15.51
5.2	9	8
5.3	9	0
5.4	9	2
Totals 6.0		10.00
6.1	36	8
6.2	27	2
Totals 7.0		6.76
7.1	9	4.76
7.2	12	2
Totals	512	78.06
Weighting	1*512	1*78.06
Metric Total	512	78.06

Table 1 Measures From OOA Example

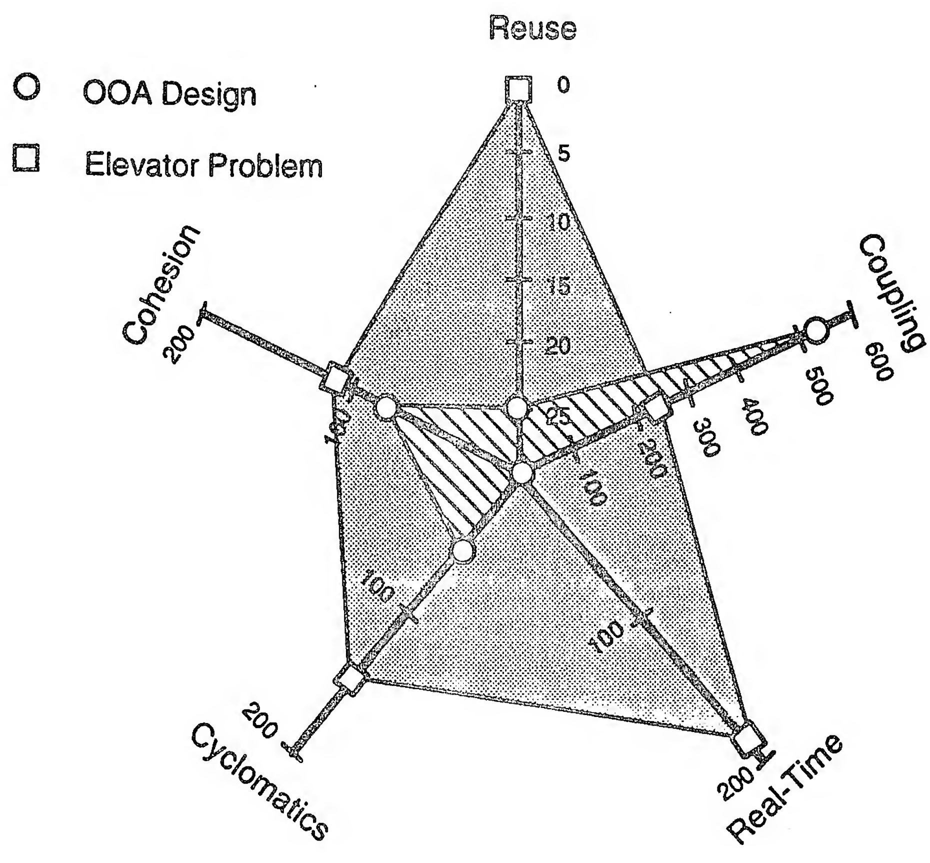
This example is more interesting because for the first time one has the ability to see real–time complexity separately from the other complexities. The REU equals 0, the coupling measure, CO, is 238, the real–time coupling, RT, is 190, and interface complexity, IC, is 149. The cohesion, COH, measure is 103.32. This model has three levels of decomposition, therefore the Depth Measure, DM, equals 0.887. The total measure is:

$$FOM = (0.887)(-0+238+190+149+103.32) = 603.44$$

Weyuker's axioms which were developed for syntactical software properties were applied to the design measure and all axioms except number seven were met. Axiom number seven deals with the order of individual statements and is implementation dependent, and therefore not applicable to a design measure. In this study of many, many measures that were researched, and yet no other evaluation criteria was so highly regarded and so universally used as Weyuker's axioms.

Example 1, the OOD problem, is markedly lower than example 2, because of the number of reused processes. Since the elevator problem had not code reuse, it is hard to properly assess the value of this measure.

Several ways to display the results of the measures have been investigated. Presenting arrays of the design measure and its components as shown in Table 1 has proven to be very useful. The arrays showing the number of occurrences of the number of each type of coupling in each process was instructive for the development of the measure and will be instructive for the designer, because it highlights the "hot spots". Figure 1 shows a Kiviat plot which has also proved very useful to compare different designs and variations of the same designs. Presentation methods will be explored during the prototyping of the user interface.



Presently no measures reflect the advantages of object–oriented designs over structured analysis designs and integrated system designs over federated designs. That is, the existing measures can not evaluate the "hidden" qualities of each these design methodologies and design approaches. More to the point, it is easier to maintain task scheduling in a federated system than in an integrated system, and therefore the inherent advantages of the integrated systems can not be shown in a measure. The measure will show that the integrated system is more complicated, not that it has advantages that are not inherit in the federated system. Yet the trend towards integrated systems is well established and important. Any handling of the evaluation of designs on structured analysis and object–oriented analysis, or different system architectures must be sensitive to the advantages and disadvantages of both, and the needs of the customer.

Conclusions

At SYMVIONICS Stephen Graff set out to find real–time software design complexity measures and discovered that design measures are methodology, language, and domain independent. Relative complexity can be measured across domains because the abstract concepts that make software complex appear in all software domains. This is true whether the software is for an embedded, real–time avionics or spacecraft systems; or a database mainframe systems. However real–time constructs must be accounted for in order to gain a complete and accurate picture of the software complexity.

The measure shows what the complexities are and where they are located. Thus at the system level, tradeoffs between competing system architectures can be made on the objective bases of cohesion, coupling and cyclomatics, and real–time complexity prior to code development. At lower levels, alternative designs can be evaluated, allowing design strengths and weaknesses to be pinpointed. Modifications to designs can be compared to earlier versions to determine whether complexity has been decreased or increase with as the design matures. Real–time complexities and thread interaction are evaluated at all levels and brought to the attention of the system and subsystem architects and designers for careful scrutiny.

Providing intuitive and judgmental information about incremental design changes is one of the weaknesses of present code measures.[8] This measure over comes that problem by providing an intuitive value for the part of the design which it is being evaluating. Furthermore, the value proportionally increases as the complexity increases.

This measure has a sound mathematical basis and was evaluated with Weyuker's Axioms which are the most widely accepted mathematical criteria for software metrics. SYMVIONICS feels that a strong mathematical basis is necessary for academic, industrial and governmental acceptance as well as necessary for soundness of the measurement concept.

Another advantage of this measure is that it evaluates the design from many points of view, at all levels of the software design, and is useful throughout the project life cycle in order to meet the needs of real–time avionics software.

Bibliography

Books

Booch, G., *Object Oriented Design with Applications,* Redwood City, CA: The Benjamin Cummings Publishing Company, Inc., 1991.

[8] Arora, V.; Greer, J. E.; and Tremblay, J. P., "A Framework For Capturing Design Rationale Using Granularity Hierarchies", *Proceedings Fifth International Workshop on Computer-Aided Software Engineering, 6-10 July, 1992,* IEEE Computer Society Press, Los Alamitos, CA, p. 246.

Cameron, J., *JSP & JSD: The Jackson Approach to Software Development,* Washington, DC: IEEE Computer Society Press, 1989.

Gomaa, H., *Software Design Methods for Concurrent and Real–Time Systems,* Reading, MA: Addison–Wesley Publishing Company, 1993.

Hatley, D. and Pirbhai, I., *Strategies for Real–Time Specification,* New York: Dorset House Publishing Company, 1988.

Kernighan, B. W. and Plauger, P. J., *Software Tools,* Reading, MA: Addison–Wesley, 1976.

Kernighan, B. W. and Plauger, P. J., *The Elements of Program Style,* New York: McGraw–Hill, 1974.

Page–Jones, M., *The Practical Guide to Structured System Design,* New York: Yourdon Press, Prentice–Hall, 1980.

Pfleeger, S. L., *Software Engineering, The Production of Quality Software,* New York: Macmillan Publishing Company, 1987.

Yourdon, E., *Managing the Structured Techniques,* Englewood Cliffs, New Jersey: Yourdon Press, Prentice–Hall, 1989.

Yourdon, E., *Modern Structured Analysis,* Englewood Cliffs, New Jersey: Yourdon Press, Prentice–Hall, 1989.

Papers

Arora, V.; Greer, J. E.; and Tremblay, J. P., "A Framework For Capturing Design Rationale Using Granularity Hierarchies", *Proceedings Fifth International Workshop on Computer–Aided Software Engineering,* (6–10 July, 1992), IEEE Computer Society Press, Los Alamitos, CA, pp. 246–251.

Henry, S. and Selig, C., "Predicting Source–Code Complexity at the Design Stage", *IEEE Software,* (March, 1990), pp. 36–44.

Honiden, S.; Kotaka, N.; and Kishimoto, Y. "Formalizing Specification Modeling in OOA", *IEEE Software,* (January, 1993), pp. 54–66.

McCabe, T. J. and Bulter, C. W., "Design Complexity Measurement and Testing", *Communications of the ACM,* Volume 32, Number 12, (December, 1989), pp. 1415–1425.

McCabe, T. J. and DuPont, E. S. W., "An Engineering Approach to Software Maintenance", *Case Outlook,* Volume 6, Number 1, (January – February, 1992), pp. 19–21.

Page–Jones, M., "Comparing Techniques by Means of Encapsulation and Connascence", *Communications of the ACM,* Volume 35, Number 9, (September, 1992), pp. 147–151.

Parnas, D. L. "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM,* Volume 12, Number 12, (December, 1972), p. 1053.

Pfleeger, S. L., "Lessons Learned in Building A Corporate Metrics Program", *IEEE Software,* (May, 1993), pp. 67–74.

Porter, A. A. and Selby, R. W., "Empirically Guided Software Development Using Metric–Based Classification Trees", *IEEE Software,* Volume 7, Number 2, (March, 1990), pp. 46–54.

Weyuker, E. J., "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering,* Volume 14, Number 9, (September, 1988), pp. 1357-1365.

TRANSITIONING SOFTWARE
FROM
DEVELOPMENT TO SUSTAINMENT:
THERE IS HELP!

Steven R. Hosner

WR-ALC/LYPSCB
380 Second Street, Suite 104
Robins AFB, GA 31098-1638

When faced with the task of moving software from a developer's facility to a government owned facility, the first impulse of many people is to run! If you were not one of those many or if you are too slow to escape, don't panic yet. There are several military documents available which can help you plan for and accomplish this task.

With the current trend being to get away from MIL-STDs, you may have some reservations about spending any significant amount of time reviewing another one. However, the major problem with MIL-STDs occurs when they are written to cover every possible situation and they are applied as is without any tailoring to take into account the actual needs of a given project. The documents described below do contain a significant amount of detail, but not so much that they overly burden someone trying to use them. Neither do they contain much information which is extraneous for almost any application nor are they overly long. Altogether, the three documents are less than 200 pages.

MIL-HDBK-347, "Mission-Critical Computer Resources Software Support" covers definitions, concepts and activities required for software support throughout a system's life-cycle. It provides standardized definitions for terms and concepts, many of which are taken from other recognized standards. It has, as its major references the 5000 series of DOD Directives, MIL-STDs 480, 483, 490, 881, 1521, 2167 and 2168. It uses concepts and definitions from the MIL-STDs, but is not so tightly bound to them that you cannot substitute other standards if needed. It defines a software support activity as "The DOD or military Service organization responsible for the software support of designated MCCR software." It provides guidelines for software support activities during all phases of a system's life-cycle. It provides guidance for how to document the programmatic basis of a software support activity and planning for software support once a system is deployed. It provides IDEF type diagrams showing post deployment software support activities. It further provides some guidance on how to transition software from development to support. Overall, this handbook provides an excellent overview of software support activities with enough detail and guidance that it is useful to most people involved in software support at any phase of a system's life-cycle.

DOD-STD-1467, "Military Standard Software Support Environment" and its companion handbook, MIL-HDBK-782, "Military Handbook Software Support Environment Acquisition", contrary to what you might expect from their titles, do NOT define a universal software support environment theoretically usable by all software support activities. That direction has been tried for various other pieces of gear and has not always been successful. Instead, these documents focus, again, on providing standardized definitions and concepts associated with software support environments. There is detailed guidance on examining the developer's environment and the deltas associated with a proposed life-cycle support environment. In addition, there is significant guidance and information on how to plan for and execute the acquisition of a software support environment.

As reference material for software support activities, these documents provide valuable insight into the activities required during each phase of a system's life-cycle and some much-needed hints on how to go about defining, acquiring and using software support environments.

FORMAL METHODS in SOFTWARE:
It's not as bad as you might think.

Steven R. Hosner

WR-ALC/LYPSCB
380 Second Street
Suite 104
Robins AFB, GA 31098

1 HISTORICAL BACKGROUND (or: How did we get here from there?)

The computer industry is rife with folklore and studies about the "software crisis" which has continued for the last thirty years. It is probably more correct to say that the computer industry has extended the crisis by evolving ever more capable hardware outstripping the new techniques and methods evolved to handle the complexity and sheer size of software packages attempted over time. It is a corollary to this situation that when systems had no software, software was no problem. When systems have only a little software, software is a little problem. When systems have a gargantuan amount of software, software is a gargantuan problem. Techniques for handling software shifted in emphasis from coding techniques, as exemplified in Dykstra's famous "GOTOs Considered Harmful" paper, to design techniques, to analysis (or specification) and design methodologies. Such shifts tended to lag the changes in hardware capabilities in the computer industry.

As the emphasis shifted, folklore and studies indicated that mistakes injected in the earlier portions of the software life-cycle tended to be the most expensive and difficult to find and fix. So, the current emphasis in the computer industry is in finding an efficient, repeatable and error-free (as much as possible) method of analyzing a computer system's requirements and designing the software for the system.

An additional problem for the computer industry is that there is a huge installed base of software which may or may not have been built using an appropriate methodology. To make matters worse, even if the installed software had been so built, unless maintenance is done with rigid discipline, over time, the entropy in software tends to increase as maintenance actions make changes to the software.

As the amount of software in systems increased, as mentioned above, new techniques and methods evolved in attempts to handle the new problems. Such techniques and methods have been evolving from non-existence to various levels of formality. John C. Napier, editor of Electronic Design News defines a formal method as: "A formal method consists of a defined set of terms and symbols that have a limited and known number of valid relationships to each other. How limited and how well-identified the relationships are establishes how useful the method will be in verifying correctness and completeness of

a specification. Mathematicians own the most formal of formal methods." [1]. Given this definition, it can be seen that a formal method is the notation and the relationships allowable between the various notational symbols.

Also in relation to this definition, it can be seen that the current state of the art in software specification and design has evolved over time. When software was not specified, specification formality was not required since specification was not done. As software became more complex, various specification and design techniques and methodologies were developed. Out of goto-less programming, structured programming evolved. Out of structured programming, structured design evolved. From structured design, structured analysis was developed. Various flavors and extensions of this methodology has occurred to handle those systems which are primarily procedural in nature and which are concerned with system control. There have been a few methodologies developed which are for systems which are primarily concerned with the storage and manipulation of data (e.g. the Jackson Development method).

From a view of both the structured analysis camp of software systems and the data oriented camp of software systems, it becomes clear that a methodology which takes into account both the procedural and the data aspects of a system will give the system developer a more comprehensive view of the system than either one alone. Coupling this with an attempt at a more direct model of the system and its environment gave rise to the object-oriented methods (Rumbaugh, Coad-Yourdon, etc.) [2],[3]. There are extensions to such methods attempting to handle the control and real-time aspects of embedded systems (Firesmith, ROOM, etc.).

All these methods rely heavily on natural language, pseudo-code (which is not heavily formal per our definition) and various graphical notations in their modeling, analysis and design. It is generally conceded that each of these methods brings new and better methods to the problem of building software systems and comprehending overall system function. However, there is no way to rigorously prove that the systems built using these methods will actually work as intended. Further, it can be very difficult to unambiguously specify exactly what it is the system is supposed to do.

2 SOME LITERATURE REFERENCES (or: What seems to be happening with formal methods?)

It is difficult to find historical information on the subject of when formal methods actually began. Doubtless, there are papers in publications such as the SIAM which dealt with the esoteric mathematical basis of formal methods prior to the 1989 examples [4], [5]. Indeed, considering the extensive use of basic mathematics of set theory, mathematical induction and deduction and the use of basic mathematical properties of integers and reals, it can be argued that the initial basis for formal methods is lost in the antiquities of time. For more concrete checks in recent times, the publication of "The Z Notation: A Reference Manual" by J. M. Spivey in 1989 [6] (please note that Z is pronounced zed) and "Systematic Software Development using VDM" (Vienna

Development Method) in 1990 by C.B. Jones [7] are certainly major landmarks in the development of formal methods and their notations. It is clear that these methods and notations were around for significant periods of time prior to the publication of these books. These books are landmarks because either shortly before or shortly after these books, the formal notations they defined (Z and VDM respectively) were standardized by the British Standards Institute into formal British standards (no pun intended). These methods are of importance because they are general purpose specification notations/languages intended to document the specification and design of a software system. While Z seems to be in more general use than VDM, VDM forms the basis for several methods and languages which generate "executable specifications" [8][9][10]. There are several projects which generate more or less popular formal notations for use as "executable specifications" [11][12][13].

Surprisingly, Sharam Hekmatpour in his book "Software Prototyping, formal methods and VDM" [14] ties together what appears to be the two extremes in software engineering, software prototyping and formal methods. Prototyping generally gives the impression of very informal techniques and a "fast and loose" approach to software development. In contrast, formal methods are, of course, highly formal, rigorous and disciplined. With tongue in cheek, you might say that of all the software development processes available, prototyping is the one most in need of enforced rigor somewhere, so it is probably appropriate that these "strange bedfellows" are mated.

As might be expected from the previous paragraph, it appears that most of the research and application of formal methods is being done in Europe as is the application of formal methods to more routine problems. It is also interesting to note that most of the more routine work done in Europe either is a derivative of the two notations/languages mentioned above or is tied directly to a given "executable specification language" which is not readily adaptable to general purpose programming [8][9][10][11][12][13][14].

There are a few notable exceptions to the general rule stated above. First is the application work done by IBM in its clean room software engineering applications in conjunction with Dr. Harlan Mills. In this case, however, the primary emphasis does not appear to be on providing a rigorous mathematical basis for the development of software (although there is certainly more mathematics involved in the process than is normal in the US). Instead, the emphasis is on providing a highly rigorous process in which the software is to be developed and the application of statistical process control to the software process to control the quality of the software so produced [15].

An exception for research is the work done by a joint group of researchers at the University of Texas and the Fujitsu Network Transmission Systems [16]. These researchers applied a formal notation and formal method to attack the problem of generating possible scenarios a new system might have to handle.

Two other research exceptions come from the Software Engineering Institute of the Carnegie Melon University. The first exception was an attempt by the SEI to specify the

behavior of an inertial navigation set using the VDM notation [17]. While this falls more into the routine uses of formal methods, considering that it is being done in the US by a US organization and that it was not expected to result in a working system, it is more closely akin to a research effort than an actual application. The other exception was an attempt to automate the translation of the Z notation into the Ada programming language using an intermediate language called ANNA (ANNotated Ada) [18]. The report showed a striking resemblance between the general philosophy and constructs of Ada and Z.

3 RATIONALE FOR FORMAL METHODS (or: Why this paper is being written.)

While the application of formal methods to the specification of massive software systems is possible, it is generally considered more practical to apply it to small critical portions of such a system's software. This allows such specifications to be rigorously examined and probed for faults while greatly improving their clarity and conciseness. If necessary, these specifications can be studied in detail and mathematically shown to be "correct" in that the software written to these specifications will do what the specification says given that the inputs to the software are within the limits given in the specification. Given the abstract reasoning power of the formal methods, it is often possible to cover a large number of cases (e.g. all integers) with a single specification.

4 A SPECIFIC FORMAL METHOD (or: This is the one I know about.)

This paper will now try to de-mystify at least one formal method, Z, by exposing some of its mathematical basis, state some of the rules by which it is constructed and show some of the specific notation used by the method.

4.1 THE MATHEMATICAL BASIS FOR Z (or: This is the sticky part.)

Z has its mathematical basis in set theory, predicate and propositional calculus and two valued boolean logic. It is now necessary (unfortunately) to provide some notation for this mathematical basis so that examples given later will be understandable. The notation will be familiar to those who have completed some college mathematics and will be kept to a minimum. Due to the limitations of the PC keyboard, the standard mathematical notation will be modified somewhat.

{a,b,c} is the set consisting of objects a, b and c

A = {a,b,c} A is the name which can be used to refer to the set
x C- A means that x is a member of set A

x C-/ A means that x is not a member of set A

P is a predicate which evaluates to either true or false (e.g. x < 2, Tuesday is a day of the week, February is a day of the week, etc.)

{x : T | P o k} denotes the set of all objects of type T which are of form k when P is true (e.g. {x : Z | x > 2 o x**2} or all integers greater than two which can be written as a square) Note: Z is a predefined type for integers in Z. (Confusing, isn't it?)

P v Q is the boolean P or Q

P AND Q is the boolean P and Q

(E D | Q o P) denotes a predicate which says there exists at least one D (a declaration such as x : Z) which satisfies both Q and P

(E1 D | Q o P) denotes a predicate which says there exists exactly one D (a declaration such as x : Z) which satisfies both Q and P

(\-/ D | Q o P) denotes the predicate which says that all D (e.g. x : Z) which satisfies Q must also satisfy P

4.2 GENERAL CONSTRUCT EXAMPLES (or: What are the pieces of a Z specification?)

This paper will address the half dozen or so constructs of Z which most HOL programmers will identify with and which can be used to construct a Z specification.

Basic Type Declarations - introduces types which have no detail to them but just represent abstract ideas which the specification must work with

* [NAME, ADDRESS, PHONE NUMBER]

Free Type Declaration - a type declaration which enumerates the possible values for the type
* REPORT :== ok | already_known | not_known

Schema Definitions - schemas can introduce the equivalent of records or define functions/procedures

```
* |-- S --------------------
  | x : A
  |-------------------------
  | P
  |-------------------------
```

denotes a schema S which introduces a variable x of type A that can only take values satisfying P, the type declaration part is called the signature of the schema and the part containing the predicate P is called the predicate part of the schema

```
* |-- S ---------------
  | x? : A
  | y! : B
  |--------------------
  | x < y AND y = x
  |--------------------
```

denotes a schema S with variable x in its signature whose value cannot be changed by S and y whose value can be changed by S

Precondition - a predicate which describes the set of states the system can be in in order to successfully complete the operation and hence satisfy the postcondition

Postcondition - of an operation is the predicate which describes the set of states the system can be in after a successful and correct attempt at performing the operation is carried out

```
* |-- S ----------
  | x? : A
  | y! : A
  |---------------
  | (x > 3 AND y = x)
  |---------------
```

denotes a schema in which x > 3 is the predicate which specifies the precondition for x and y = 3 is the postcondition for the schema

* Λ (capital delta)
  prefixed to a schema name to introduce a set of before and after states for the schema (i.e. the data declared in the schema can be changed)

* _= (capital xi)
  prefixed to a schema name to introduce a set of before and after states such that the after states are equal to the before states (i.e. the data declared in the schema cannot be changed)

Global Declarations - these declarations introduce types or constraints which will be visible to the rest of the specification

```
* | x : A
  |------------------
  | P
```

denotes the declaration of a global variable x of type A which is optionally constrained by the predicate P

```
* P
  denotes a global constraint P put on a system model
(e.g. MAX_NUM < 100)
```

Generic Schemas - schemas which contain local basic type declarations known as formal generic parameters which can later be assigned with the actual types to be used, the actual generic parameters

Example:

```
* |== SQUARE [A,A] ==================
  | x?,y? : A
  | z! : A
  |----------------------------
  | (\-/ x,y : A; o (x > y AND z = y*y) v (x<y AND z = x*x)
  |----------------------------
```

SQUARE is a function which sets the output equal to the square of one of the inputs based on which input is the larger of the two

4.2 GENERAL RULES (or: How a Z specification is put together.)

The general structure of a Z specification is similar to that of most High Order Language (HOL) programs, variables are declared and those variables are manipulated. In both HOLs and Z, variables must be declared prior to their use and they must be assigned a type. Both have the concept of "scope" of a variable. That is, a variable is known only to those portions of a HOL program (and a Z specification) which occur after the variable has been declared. Variables declared within certain constructs (e.g. a procedure in a HOL) are known only to that construct and not to other constructs. HOLs have record structures and procedures or functions. In Z, there are schemas which serve as either a kind of a record or a procedure. Both also expect comments to be included to help clarify them. The primary difference between Z and HOLs is that Z is not intended to become executable on a machine and Z focuses primarily on specifying WHAT the program is to accomplish, not how it is to be done.

4.2.1 EXAMPLE Z SPECIFICATIONS (or: This is how you use the notation)

This example is a modified example of a computerized birthday book from the Spivy book [6].

This is the Z specification for the Birthday Book example from Dr. Spivey's book, "The Z notation : a reference manual".

This introduces the basic types NAME and DATE. Note that no assumptions are made about the implementation of NAME and DATE, only that the abstract concept of the two types exists.

[NAME, DATE]

REPORT is a "free type" which defines a new type to be used later
REPORT :== ok | already_known | not_known

The schema BirthdayBook introduces the state space of the specification. known is the set of names with birthdays recorded. birthday is a partial function (notation -|->) which relates names to dates. dom is a function which returns the domain of a function

```
|-- BirthdayBook ---------------------
| known : P NAME
| birthday : NAME -|-> DATE
|---------------------------------------
| known = dom birthday
|---------------------------------------
```

InitBirthdayBook initializes the system state to no names known and so no dates are associated with them.

```
|-- InitBirthday ----------------------
| BirthdayBook
|-------------------------------------
| known = { }
|-------------------------------------
```

RAddBirthday will perform the function of adding a new name and date to the book provided that the name is not already in the book. It will also handle the situation when a name is already known.

```
|-- RAddBirthday ----------------
| /\BirthdayBook
| name? : NAME
| date? : DATE
| result! : REPORT
|------------------------------
| (name? C-/ known AND
|    birthday' = birthday U {name? |-> date?} AND
|    result! = ok) v
| (name? C- known AND
|    birthday' = birthday AND
|    result! = already_known)
|------------------------------
```

RFindBirthday will perform the function of finding the date
associated with a name and handle the situation when the name is not known.

```
|-- RFindBirthday ------------
| _=BirthdayBook
| name? : NAME
| date! : DATE
| result! : REPORT
|------------------------------
| (name? C- known AND
|    date! = birthday(name?) AND
|    result! = ok) v
| (name? C-/ known AND
|    result! = not_known)
|------------------------------
```

Remind is a function which returns a list of names which all have a
birthday on the date input into the function.

```
|-- RRemind --------------------
| _=BirthdayBook
| today? : DATE
| cards! : P NAME
| result! : REPORT
|------------------------------
| cards! = {n : known | birthday(n) = today?} AND
| result! = ok
|------------------------------
```

5 SUMMARY (or: Finally, the guy is done!)

Notice in all this specification, the use of mathematical notation and concepts allowed the requirements to be unambiguously defined without constraining the implementation. It is this abstract precision and clarity which lends formal methods to the specification of software. The examples given are only a part of the capabilities of Z and are simply meant to get across the idea that formal methods, while different from the specification methods usually used, need not be so esoteric that they are incomprehensible. For further reading, the Imperato book [19] is a gentle introduction to Z. The Diller book [20] is a somewhat more rigorous book, but still manageable for most people who have had some set theory and a general background in programming.

LIST OF REFERENCES

[1] Napier, John C. "Nail Down Your Spec With Formal Methods", Electronic Design News, pp. 74-82, December 24, 1992

[2] Rumbaugh/Blaha/Premerlani/Eddy/Lorensen, (1991), Object-Oriented Modeling and Design, Prentice Hall, ISBN 0-13-629841-9

[3] Coad/Yourdon, (1991), Object-Oriented Analysis, Yourdon Press Computing Series, ISBN0-13-629981-4

[4] Galil/Haber/Yung, "Minimum-Knowlege Interactive Proofs for Decision Problems", Society for Industrial and Applied Mathematics, Vol. 18, No. 4, pp. 711-739, August 1989

[5] Goldwasser/Micali/Rackoff, "The Knowledge Complexity of Interactive Proof Systems", Society for Industrial and Applied Mathematics, Vol. 18, No. 1, pp. 186-208, February 1989

[6] Spivey, J. M. (1989), The Z notation, Prentice Hall International, ISBN 0-13-983768-X

[7] Jones, C. B. (1990), Systematic software development using VDM, Prentice Hall, ISBN unknown

[8] Borba/Meira, "From VDM Specifications to Functional Prototypes", Journal of Systems and Software (New York, Elsevier North Holland), Vol. 21, No. 3, pp. 267-278, June 1993

[9] Fields/Elvan-Goransson, "A VDM Case Study in mural", IEEE Transactions on Software Engineering, Vol. 18, No. 4, pp. 279-295, April 1992

[10] Andersen/Elmstrom/Lassen/Larsen, "Making Specifications Executable - Using IPTES Meta-IV", Microprocessing and Microprogramming (Amsterdam, North-Holland Publishing Company), Vol. 35, No. 1-5, pp. 521-528, September 1992

[11] van Hee/Hildebrand/Copelli, "PROOFS: Application Engineering Based on Formal Methods", Microprocessing and Microprogramming (Amsterdam, North-Holland Publishing Company), Vol. 35, No. 1-5, pp. 29-37, September 1992

[12] Cuypers, Ludo, "Automated Implementations of Lotos Specifications", Microprocessing and Microprogramming (Amsterdam, North-Holland Publishing Company), Vol. 35, No. 1-5, pp. 729-735, September 1992

[13] Dauphin, Michel, "SPECS: Formal Methods and Techniques for Telecommunications Software Development", Microprocessing and Microprogramming (Amsterdam, North-Holland Publishing Company), Vol. 35, No. 1-5, pp. 117-124, September 1992

[14] Hekmatpur, Sharam & Ince, Darrel (1988), Software Prototyping, Formal Methods and VDM", Addison-Wesley, ISBN unknown

[15] Linger, Richard C., "Cleanroom Process Model", IEEE Software, Vol. and No. unknown, pp. 50-58, March 1994

[16] Hsia/Samuel/Gao/Kung/Toyoshima/Chen, "Formal Approach to Scenario Analysis", IEEE Software, Vol. and No. unknown, pp. 33-41, March 1994

[17] Pederson/Klein, "Using the Vienna Development Method (VDM) To Formalize a Communication Protocol", Technical Report CMU/SEI-88-TR-88-26, November 1988

[18] Place/Wood/Luckham/Mann/Sankar, "Formal Development of Ada Programs Using Z and Anna: A Case Study", Technical Report CMU/SEI-91-TR-1, February 1991

[19] Imperato, Michael (1991), An Introduction to Z, Chartwell-Bratt (Publishing and Training) Ltd, ISBN 0-86238-289-0

[20] Diller, Antoni (1991), An Introduction to Formal Methods, John Wiley & Sons, ISBN 0-471-9248-X

Data Integrity Processes -
A Practical Software Implemented Fault Tolerance Methodology

Robert J. Kreutzfeld[1]
Dr. Richard E. Neese[2]
Marc Pitarys[3]

Abstract

As computing capabilities continue to improve, there will be a rise in the number of computational faults caused by more complex software, by the increased number of untested software states, and by a wider incidence of hardware faults stemming from increasing hardware speed and density. For non-life critical systems, the extensive use of current software-implemented fault tolerance schemes (N-version and recovery blocks) to increase the system's reliability would be cost prohibitive due to the inherent complexity of properly implementing them.

A new cost-effective methodology, known as Data Integrity Processes (DIPs), has been developed and is presented. This simple and practical approach will be compared to the current methods of N-version, recovery blocks, and executable assertions. The DIP development methodology is also discussed, with an emphasis towards Ada and avionics. Also highlighted is the progress being made on two active DIP projects.

Introduction

Visualize a fighter aircraft flying straight and level over the desert. For no apparent reason, the range output of the fire control radar drops precipitously from 12,000 feet to 10 feet in approximately two seconds. How can the potential for obviously erroneous data such as this be anticipated and protected against? Or what if the data validity bit in the radar altimeter is set to valid a fourth of a second before the good data is ready in the output buffer, resulting propagation of corrupted data to subsequent calculations. Is there a means whereby this sort of data can be easily detected and prevented from propagating throughout the system? Can expert knowledge be implemented into a few rules capable of detecting these "obvious" anomalies without impacting real-time performance and be affordable?

The solution to both of these anomalies, which were encountered during actual flight testing of a research aircraft, can be found in the realm of software implemented fault tolerance. There are at least three good methodologies existing today that can be used in any software product to improve its reliability; the two best known methods are N-version and recovery blocks. N-version is characterized by more than one version of the application program being developed (usually development is independent, but from the same specification); all N versions are then executed and the results compared. The recovery block methodology, on the other hand, usually consists of one main version followed by an acceptance test. If the acceptance test fails, a backup version is executed to produce the needed output. The third methodology through which software fault tolerance can be

[1] TASC, Inc., 2555 University Blvd, Fairborn, OH, 45324 rjkreutzfeld@tasc.com
[2] TASC, Inc., 601 Russell Parkway, Warner Robins, GA, 31088 reneese@tasc.com
[3] Dept. of the Air Force, WL/AAAF, 2185 Avionics Circle, Bld 635, Wright-Patterson AFB OH, 45433-7301 pitarysmj@aa.wpafb.af.mil

implemented is that of executable assertions. Executable assertions are essentially embedded acceptance tests that evaluate the "reasonableness" of data in terms of its adherence to an expected range of values.

The trend in computer applications is towards increasing capability that requires larger and more complex computing systems (both hardware and software). Reliable operations of these capabilities are more important now then ever. That is, while not life critical, these mission critical capabilities need to be as reliable as economically feasible.

The cost associated with implementing the typical and expensive fault tolerance schemes (such as N-version and recovery blocks) into life-critical avionics, such as digital flight control computers, has always been and will be justifiable. However, inclusion of these typical fault tolerance schemes into mission critical avionics has a difficulty in justification. Non-life-critical avionics has, in the past, only relied upon very limited forms of fault tolerance capability. Some existing avionics computer software products, for example, only check for data validity when a data item appears from an external interface and assumes from that point on, that the data is valid. This data check is usually accomplished by examining accompanying data validity bits and checking for data range. Experience has shown that this is often an inadequate means of detecting data errors and limiting their propagation due to the fact that a good validity bit does not necessarily mean that the data is good. In addition, the data is susceptible to corruption following arrival. Nonetheless, this cursory verification process has been relied upon because of limitations in the embedded computer's memory and processing power. As computer capabilities increase, inclusion of more robust forms of fault tolerance becomes more feasible. There is, therefore, a demonstrated need for a software fault tolerance method that can be employed to mission critical applications that is less expensive to develop than N-version or recovery blocks, and yet effective enough to warrant the effort and cost of implementation.

Data Integrity Processes (DIPs)

To achieve this goal of lowering costs without losing effectiveness, a scheme has been developed utilizing reasonableness tests with simple tolerance mechanisms on unit input and output parameters. The tests make use of common sense rules and utilize many other system parameters. Using these rules, functions can be built that first detect data "unreasonableness" and then provide some degree of tolerance. These input/output functions, implemented as separate program procedures around the application program, are called Data Integrity Processes (DIPs) and form the foundation of a cost-effective software fault tolerance methodology.

DIPs are software-implemented error detection and tolerance functions located before (DIPin) and/or after (DIPout) the calling of an application function. Their detection task is accomplished by scrutinizing the data using a framework of rules to erect the tightest possible constraints on data acceptability. (This task can vary in complexity as DIPs can operate as simple range or interframe rate checks, or they can be built to apply statistical trend analyses or many if-then rules.) When the results fall outside the acceptable range, the tolerance aspect of the DIP's dual role is triggered. Possible response techniques include simply using the last valid data, extrapolating, or resorting to another algorithm to recompute the data.

To demonstrate the application of DIPs, an actual example will be considered. After rehosting an Air-to-Ground (AG) bombing algorithm to Ada, simulated runs were performed while recording the time_to_release output. This output is normally a smoothly declining value that, upon reaching zero, signals a weapons release. Assume that an error of some type (e.g., hardware glitch not detected by parity check, overwrite, etc.) caused vertical acceleration, one input to the AG algorithm, to change as shown in Figure 1(a). This spike would have an impact on the AG algorithm's time_to_release output such as that shown in Figure 1(b), with the result being a mission failure, i.e., the weapon being released about one second early. A DIPin was then implemented on the vertical acceleration value to check for excessive acceleration rates; if such an excess was detected, the DIP would simply use the last good value. (The value set as the "limit" was based on an estimate of the aircraft's maximum acceleration rate.) Figure 1(c) demonstrates the DIP's ability to ameliorate the acceleration spike, enabling the time_to_release value to be maintained above zero seconds as shown in Figure 1(d).

As an alternative, a DIPout can be placed on the time_to_release value to limit value jumps to 0.25 seconds during the last 10 seconds before release. If a large jump is detected, the DIP calculates a new value by

extrapolating from the last 10 good values. Using the scenario in the beginning of this example, a DIPout would suppress the early release caused by the acceleration spike, as illustrated in Figure 1(e). Similarly, any severe intermittent error emanating from the AG algorithm due to either a hardware or software glitch would be tolerated by this very simple DIPout.

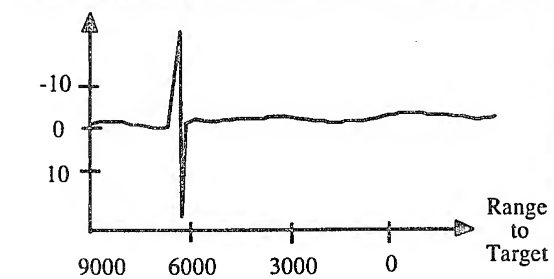


Figure 1(a). Intentional spike on acceleration

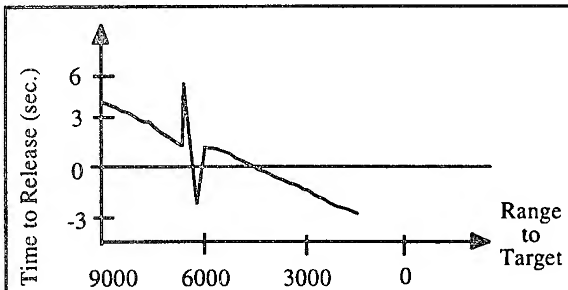


Figure 1(b). Time to release showing early release

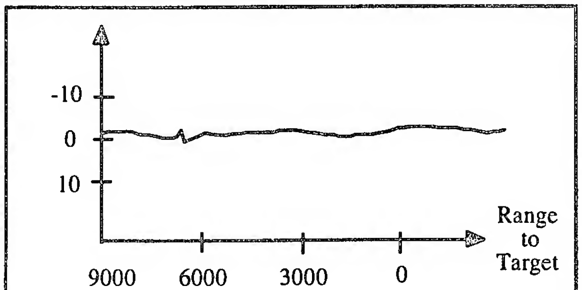


Figure 1(c). DIPin using last good value

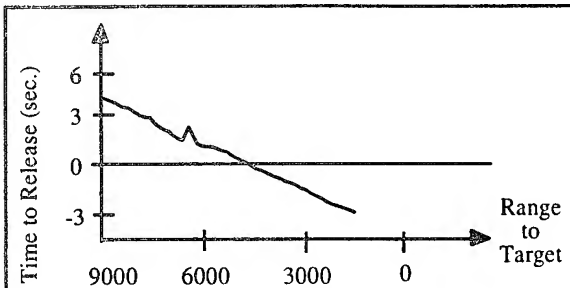


Figure 1(d). Time to release after using DIPin

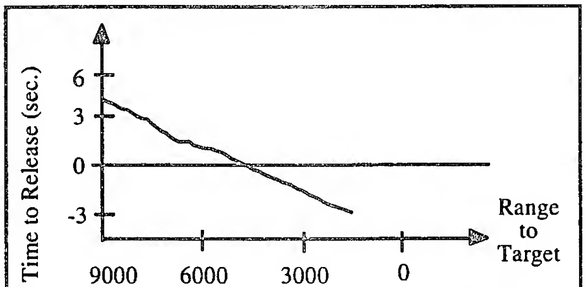


Figure 1(e). Time to release after using DIPout

Because DIPs do not necessarily verify that data is accurate (but rather are limited to determining whether or not it is "reasonable"), they should be designed to use as much support data as needed to make good common sense decisions. Furthermore, knowledge of algorithm behavior and sensitivity is essential to the creation of well-tailored DIPs. With an understanding of the way the algorithm behaves or is supposed to behave, DIPs can be written to expect this behavior. Having done this, the proper operating region of the algorithm can be constrained, with anything falling outside the specified area assumed to be "bad" data.

Unfortunately, while an acceptable operating region can be specified, this does not guarantee that all data meeting the established criteria is really "good" data. Like N-version and recovery blocks, DIPs are not intended to be foolproof and bad data can still be deemed proper so long as it falls within the stipulated "good" data space. For example, all values of 16 bit twos-complement representation Binary Angular Measure (BAM) values are equally valid, but DIPs could sense inordinately abrupt jumps in values for roll, pitch, azimuth, etc. This lack of absolute certainty is not a major drawback, however, in that the primary function of the DIPs is to catch instances

of gross variation in data values rather than gradual data drifts. To accomplish this primary function, DIPs may be designed to reject instantaneous changes in the data (magnitude, magnitude rate, etc.) as well as short-term changes over time (e.g. averages), with appropriateness to be determined based upon several factors (known data and algorithm behavior, mission state, etc.). For example, to return to the scenario outlined in the introduction, suppose the range output of a fire control radar plunged from 12,000 feet to 10 feet in two seconds while the aircraft was flying straight and level over flat terrain. As in the case of the bomb drop example, by augmenting the rules with common sense, an understanding of expected algorithm behavior, and other system parameters (e.g., flat terrain expected), DIPs can be written to recognize the "unreasonableness" of such a jump and thereby ensure that such data is summarily ignored and error propagation consequently limited. This is an enhancement of embedded "performance monitor" software found in some fielded inertial navigation systems (INSs).

The key to the DIP method is the utilization of system and mission data. This is a core distinction between DIPs and other software fault tolerance methodologies in that the latter rely solely on the data incoming to a specific unit with no reference to the "big" picture. As a result of this microscopic focus, data shifts that a human would recognize as clearly absurd go undetected because the fault tolerance scheme only sees them as being within the overall acceptable range. To illustrate, mission planning typically includes information on weapons stores, destination, targets, etc., but doesn't necessarily specify "incidentals" such as whether the mission will be flown over land or water. Including such "incidentals," however, would enable some parameters to be more tightly restricted on the basis of purely practical realities. If, for example, the mission were to be flown over water and this parameter were included, then the DIPs responsible for checking radar altimeter altitude could be more tightly constrained because of the inherently restricted range of acceptable variation. Digitized land information would have even wider application and would prove invaluable in constraining not only altitude determination, but radar data and targeting information as well. In summary, the more mission information detail and system level knowledge that can be provided to constrain the DIPs, the more it will appear as if they are exercising "common sense" and the more effective they will be in correctly identifying data as good or bad. In fact, this concept of "missionized" data and software functions is of paramount importance in the software architecture of the next generation strike fighter.

While DIPs are written to detect and circumvent "bad" data stemming from latent hardware/software errors, it must be emphasized that they are not intended to supplant normal developmental verification, validation, and testing processes. Since these phases are typically constrained by limited time and budget, they should not be further abbreviated simply because DIPs may be added. One reason is that it would be difficult to determine which specific tests the DIPs could effectively replace. Secondly, a good product with little embedded fault tolerance is the expected norm for the initial release of non-life critical applications, meaning thorough testing will be required regardless. Rather, the role of DIPs will be to improve the software's reliability during hardware glitches and/or untested software states, both of which will persist to varying degrees throughout the product's lifecycle.

With this introduction to DIPs, it is no doubt clear that they share a number of similarities with existing software-implemented fault tolerance methods. Given their reliance on acceptance-test-type data validation, DIPs could be categorized as a recovery blocks methodology; for the same reason, and because of their possible duplication of application code logic, they could be categorized as N-version. Even more logically, DIPs could be categorized as a subset of the executable assertions methodology. Therefore, to illustrate the distinctiveness of these approaches, shared capabilities as well as significant differences found with N-version and recovery blocks will be briefly discussed and summarized in the following paragraphs, followed by a closer examination of the DIP's relationship to its closest cousin, the executable assertion.

DIPs and N-Version

The N-version method of fault tolerance is based on multiple computations which can be replicated in time, space and/or information [1]. Because of this multiplication of effort, effective N-version applications require a significant code development effort. They also permit different interpretations of the specification which, if incorrect, can result in erroneous data and mandates the use of a reliable decision algorithm to determine which of several results should be used.

Because it is based on the duplication of results, N-version is best able to determine whether the computed result is actually correct (see Figure 2). DIPs, by contrast, require only one version but, because they evaluate on a scale of "reasonableness," are less effective in determining data accuracy. There is a caveat, however, in that most non-life-critical functions have many non-redundant inputs (especially sensor data). This means that in an N-version application, as illustrated in Figure 2, if the incoming data is corrupted, computing an outcome three times won't identify the error and halt its propagation. DIPs, on the other hand, attempt to limit error propagation by recognizing the invalidity of both incoming and outgoing data through the imposition of "reasonableness" checks.

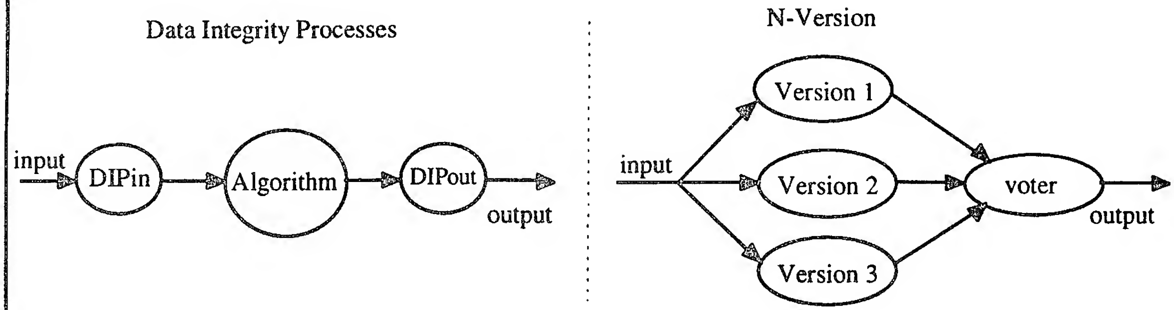


Figure 2. DIPs vs. N-Version

These differences aside, however, DIPs may sometimes be classified as an N-version methodology in that the rules relied on by the DIPs, which are based on the specification, may duplicate the underlying functions, i.e., duplicate logic used as a cross-check may be employed to detect erroneous data. This is especially true of data which takes on few values. For example, a mode command word for a fire control radar takes on many states, such as air-to-ground track, standby, etc. Strict rules govern how these states transition during normal mission operation. For a DIP to have the knowledge of what constitutes proper transitions, the DIP developer must implement the same rules into the DIP.

DIPs and Recovery Blocks

As depicted in Figure 3, the recovery blocks method is very similar to the DIP methodology, imposing an acceptance test on the data output by the primary unit. (For a good overall discussion of recovery blocks, see reference [2].) Using recovery blocks, if the data fails this test, a secondary unit recalculates the original computation, effectively serving as an N-version but in sequence rather than simultaneously. However, while developed in an N-version style, the secondary unit is usually a simpler algorithm that requires less code development and is therefore less costly to implement than N-version.

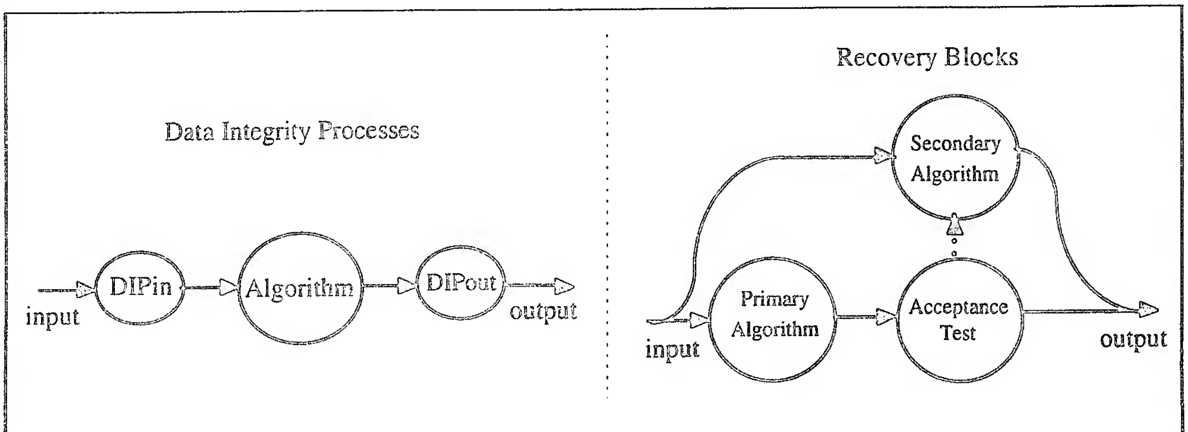


Figure 3. DIPs vs. Recovery Blocks

Like DIPs, recovery blocks have difficulty assessing the correctness of data. Furthermore, initializing the secondary unit to the last known good state can become a complex task. DIPs represent a simpler means of identifying bad data because they experience no such "back tracking" problem; unlike recovery blocks, the DIP methodology does not include recovery as a necessary step. Using DIPs, it is sufficient to recognize that the most recent data is "bad" and that either the last good data or an extrapolation should be used instead.

The remaining difference between recovery blocks and DIPs is that DIPs can scrutinize input data. When using recovery blocks, the generating algorithm may not check input data to the degree required. If, for example, the data had been corrupted by hardware glitches or data transmission problems, or even been overwritten by another unit, neither recovery blocks nor N-version would recognize that the data was unreliable. DIPins fill this blind spot, providing an additional level in data integrity protection.

Preliminary Comparison of DIPs, N-Version and Recovery Blocks

Figure 4 depicts a life cycle cost comparison of N-version (for three versions), and recovery blocks (with one recovery block) as determined by Laprie, et al. [3] N-version shows an average cost of 2.25 times that of not using any fault tolerance, while recovery blocks entails a cost of approximately 1.75 times. DIP costs will range from just above 1.0 (for DIPs incorporating simple detection and recovery schemes) up to the same cost as recovery blocks (for the DIPs which duplicate functionality).

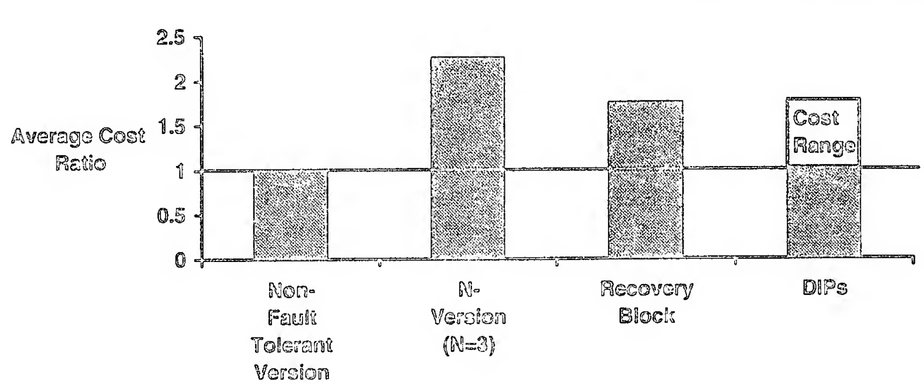


Figure 4 Life Cycle Cost Comparison

In addition, N-version and recovery blocks costs will rise as complexity of the application increases. DIPs will not because in general they only treat the reasonableness of the data behavior and provide simple tolerance mechanisms. Most of the DIP cost will be in defining the specifications. This includes analyzing the applications behavior, sensitivity, mission importance, etc., tasks which are normally performed anyway. The amount of software which would have to be developed is intended to be small (and reusable).

Comparing the error detection capabilities, N-version should be the best method provided that all N-version developers work with proper system specifications and interpret them correctly. If these prerequisites are met, the sheer redundancy of multiple versions developed by independent teams assures very good error detection capability. Note that if the input to the redundant algorithms is itself not redundant (e.g., single sensor, or data generated by a single algorithm), then the error detection capability is only better for errors which occurred during algorithmic computation. Recovery blocks and DIPs, on the other hand, should demonstrate roughly equivalent performances since both use a single acceptance test for detecting the errors (see Figure 3). Their detection capability is only as good as the acceptance tests.

For comparing error tolerance, the quality of the DIPs' performance depends in large part upon the duration of the error and the capability of the tolerance mechanism employed. When comparing the relative abilities of the three methodologies to tolerate short intermittent errors, all three are equally capable, provided the error is gross enough to be detected by the acceptance tests. During the tolerance period, simple tolerance mechanisms such as using the old value or extrapolation, have limited effectiveness over time due to the highly dynamic nature of a flight environment. N-version and recovery blocks, on the other hand, tolerate errors over an extended period of time quite well since both incorporate adequate backup processing.

The final comparison is the degree to which each methodology improves the overall system's ability to tolerate intermittent faults. This comparison is derived from the average of the performance categories above, offset by a pervasiveness-of-use factor. The result is that, while N-version may be the most effective technique in a one-on-one application, DIPs should be the overall leader because their significantly lower cost makes it feasible to incorporate them widely throughout the system. DIPs can be placed at the input and output points of most units within a software application and, through this expansive coverage, provide a broader blanket of protection per incremental development (or maintenance) dollar than either N-version or recovery blocks.

DIPs and Executable Assertions

An existing alternative to N-version and recovery blocks, and a class within which DIPs can be categorized, is the executable assertions methodology. Executable assertions are program statements embedded into the actual code of the functions. They have been used to facilitate program verification [4] [5], test and debug [6] [7] [8], documentation, and software fault tolerance [2] [9]. Executable assertions can be placed anywhere, but those that check the reasonableness of input data are called entry assertions; those that check the reasonableness of output data following unit computation are called exit assertions.

Manmood, et al., [8] illustrates how assertions can be applied to testing life-critical flight control software using three types of assertions. The first is based on the range of variables (e.g., ABS(ROLL)<0.165), the second is based on the inverse relationship of one variable to another, and the third shows assertions which check the state of variables. All three of these simple assertions utilize only the data of interest. As previously discussed, DIPs can be as simple as this, yet should be as intelligent as possible, making use of not only the data being scrutinized, but other system parameters as well.

Despite the fact that executable assertions have been in existence for many years and are a relatively simple concept, they have been put to very little formal use, especially in non-life-critical applications. Industry reticence could initially be explained by the limited power of the general purpose computers used; these machines were basically incapable of handling any more than their key functions. But while recent advances in memory capacity and processing power have rendered the hardware more than capable of supporting assertions, assertions are still not incorporated to any significant degree. This can best be explained by the fact that, even though hardware is becoming more capable, software development costs are continually rising with no real near-term solution. As a result, program managers are faced with the task of weighing the value of incorporating software

fault tolerance and justifying that cost, a difficult task given the unquantifiable nature of the benefits derived from fault tolerant software. This problem is exacerbated if the product is to be initially developed with the assertions already in place since this will only increase the difficulty inherent in producing a viable software product within the first release schedule. Research into a broader class of life-cycle cost metrics associated with different types of fault-tolerance mechanisms would enable enhanced cost estimation programs, calibrated across several programs.

Another impediment to the incorporation of executable assertions may be the lack of specificity in how they may best be used, i.e., since they are a general concept and encompass so many variations in structure, it is frequently unclear how to best apply them to a particular piece of software. The typical practice when using assertions is to embed them into a unit, but this causes significant clutter. To remove this clutter, the language in use could be extended to support assertions or the assertions could exist as individual, called procedures. The DIP methodology proposes to pull the assertions out of the application unit altogether and limit them to external entry and exit assertion procedures. This latter method, along with the expanded use of other system data, is what makes DIPs unique. It is this uniqueness, coupled with increasing processor power and memory, that makes this approach of software implemented fault tolerance practical, useful and effective.

Limiting DIPs by definition to input and output parameters facilitates proper software development, provides better configuration control, and yields a higher quality product after the extra software is added. DIPs can be added subsequent to initial unit development and, by eliminating any need to embed fault tolerance within nearly all of the actual application units, avoids the undesirable software engineering practice of changing finished and tested application units, and all the risks associated with this approach. As a result, DIPs will have minimal impact on the application code that has already been developed and tested. (Although the whole product would be retested after DIPs are added, existing application unit test procedures for which the DIPs are intended would not have to be modified.) In addition, because of the practicalities of available development time, the elimination of any need to incorporate them in the first product means DIPs are likely to produce results superior to those which can be achieved with executable assertions. Since they often comprise part of the rush to develop unit software on schedule, the more general class of executable assertions is likely to receive limited attention. DIPs, by contrast, can be developed in parallel with the first release, with full attention to the function they are intended to perform (without the associated programmatic pressure of achieving operating capability at the same time as the first release's deadline).

By serving to make reasonableness checks easier to implement and more intelligent, DIPs may be able to overcome industry reluctance to incorporate software fault tolerance techniques into non-life critical applications. In addition, DIPs can be added using any language, without necessary extensions. The only requirement is possessing the extra processing power and memory necessary to execute this additional code.

The DIP Development Methodology

To realize their potential, DIPs must be fully incorporated into the software development cycle. The question then is how to best integrate the DIP development methodology into an existing software development process to maximize usage while minimizing implementation difficulties and cost.

Good software development is a disciplined activity which, on large projects, is divided into tasks that are subsequently performed by different groups. A typical development structure flow is depicted in Figure 6. To begin, there is generally a systems level group composed of individuals who determine the system design and requirements from the user's specifications. A mechanization group is responsible for actually creating the detailed design of the system ("mechanization" is a term of art to describe the process of generating software detailed design documents from the system requirements) and is sometimes supported by an algorithm development group. Finally, there is a group responsible for the implementation and test of the design, the software implementation group. Because each of these groups approaches the task from a slightly different perspective, there are benefits to be derived from having each group design DIPs. The system group, for example, is fully conversant with the specifications of the external systems and has a more comprehensive understanding of the mission. The mechanization and algorithm groups, on the other hand, have a thorough grasp of the algorithms to be implemented. Either the system group or the mechanization/algorithm groups may be more or less lenient

on any given data parameter depending on that parameter's relationship to the ultimate directive. By allowing each of these groups to design DIPs, these differences in perspective will yield both cross checking and a wider range of coverage. Figure 7 depicts an example of where DIPs might be installed in a software product and who would be responsible for their generation.

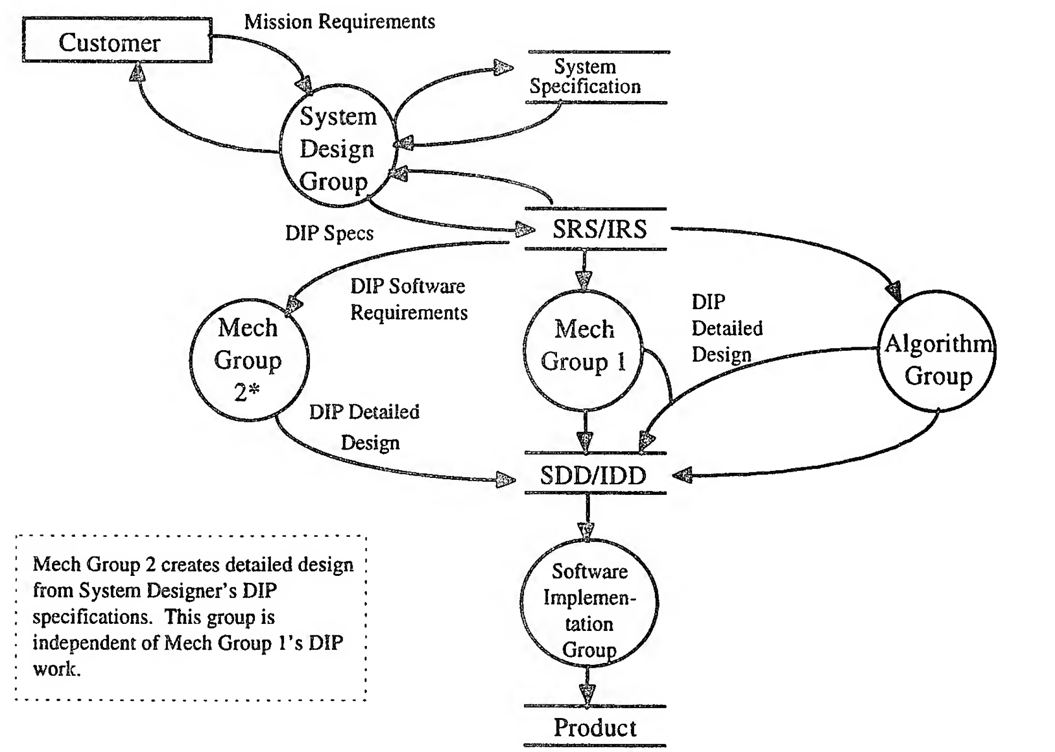


Figure 6. A Possible DIP Development Environment

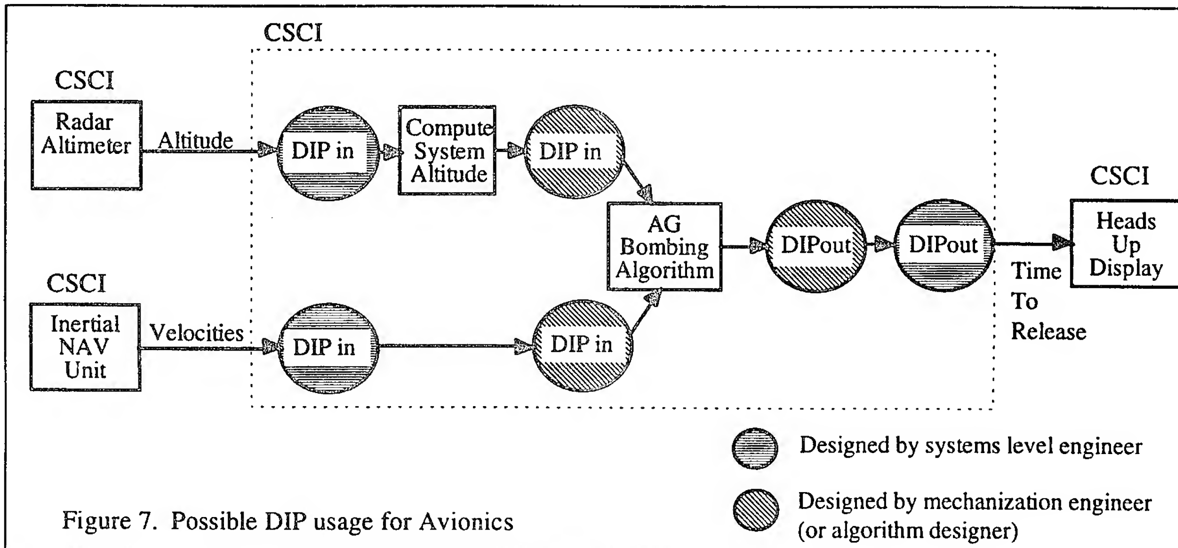


Figure 7. Possible DIP usage for Avionics

In summary, the industry's current software development processes, based on a structured development process such as DOD-STD-2167, are capable of supporting a proper DIP design and development environment with little change to the process itself.

DIPs and Ada

Ada has several unique features that incorporate some of the capabilities characteristic of software fault tolerant methodologies [10]. For example, using Ada one can implement executable assertions using the Raise statement and Exception blocks. Extra software can be written for the acceptance test of the data and, if the data should fail, the exception block can be enabled to perform fault tolerance.

Any good Ada unit design will include one or more exception handlers to allow for more robust operation. The complexity of these units depends mostly on the degree of significance placed on software robustness. Minimal use of exception handlers is anticipated, with error reporting being the primary function.

Ada also implements other forms of run-time data checking. For example, as part of constraint checking, ranges of variables can be checked and if an out-of-bounds data item is detected, the first level of exception handling is executed. These checks act as simple executable assertions as well as simple DIPs, but more elaborate checks must be performed to evaluate the "reasonableness" of the data. However, adding even these simple exception handling lines of code to an existing module can complicate the initial product in terms of quantity of code and readability, both of which can create and/or hide other errors within the application code, and both of which the DIP methodology was developed to avoid.

To make the most of Ada's run time checks when developing software to which DIPs will be added, we propose to just implement a "when OTHERS" exception block at the end of every module. This catch-all exception block would be executed if anything in the module should fail, e.g., constraint check, divide by zero, floating point overflow, etc. Because of its catch-all nature, the exact source of the error would not necessarily be known. So, when anticipating the addition of DIPs, the application unit's responsibility can be limited to setting a flag to indicate that a failure occurred and that at least some of the output data was corrupted. This flag would then be used by the DIPout as an indication that all DIP tests should be bypassed and all tolerance mechanisms imposed immediately. Obviously, this is the easiest and least complicated method of adding DIPs. A slightly more elaborate alternative would be to identify which output was corrupted, thereby allowing for only one forced tolerance function to be executed. The advantage to this is derived from the DIP philosophy that it is better to have a reasonable new value than a value generated by a simple tolerance scheme.

Automatic DIP Generation

As stressed earlier, Ada's run-time checks represent a simple form of "reasonableness" checks but are simply unable to provide complete coverage [10]. It has been proposed that a meta language be used as part of the code product [7], in which case the meta language would have to be preprocessed and converted into Ada source code. DIPs, on the other hand, wouldn't require a meta language or any language extension. Rather, DIP procedures can be written directly in the application's language. A chief advantage to this is the significant opportunity this presents for software reuse. For many of the DIP functions, the only changes that would be required would be threshold parameters or condition variables. As a result, DIP construction can become a much more automatic process, wherein the user specifies the type of DIPs, the thresholds, the tolerance scheme, any other conditional rules, and the structure of the DIP (which rule first, etc.). This information could then be used in a code generator tool which would use existing Ada code of DIP functions, and would get the appropriate information out of a CASE data dictionary (or Ada package specifications). With the addition of some simple control flow, a complete DIP unit would be produced from already existing software, thereby significantly decreasing the costs associated with software development.

Conclusion

With hardware density and software complexity increasing, the number of hardware glitches and faults from untested software states will only continue to rise. The "proper" use of current software fault tolerant techniques is very costly and for this reason will continue to be avoided in non-life-critical applications. We showed that the DIP methodology represents an available compromise which can effectively tolerate short, intermittent errors. This methodology is compatible with current software development practices and does not necessarily increase the initial cost of the product. Because they exist only as separate entry and exit procedures, DIPs don't clutter the application code as embedded executable assertions can and therefore do not open the door to errors stemming from increased complexity. Finally, because DIPs can be more easily added at any time, they don't have to complicate the initial software release schedule, thus allowing total focus on the application's timely completion. Based on all these considerations, it is the premise that through the pervasive incorporation of DIP software fault tolerance, hardware glitches and previously unidentified software errors can be detected and circumvented. This strategy will ultimately net increased product reliability, potentially with concomitant life cycle cost reductions.

References

[1] Algirdas Arizienis, "The N-Version Approach to Fault-Tolerant Software," IEEE Trans. on Soft. Eng., Vol. SE-11, No. 12, pp. 1491-1501, Dec. 1985.

[2] Brian Randell, "System Structure for Software Fault Tolerance," IEEE Trans. on Soft. Eng., Vol. SE-1, No. 2, pp. 220-232, June 1975.

[3] J.C. Laprie, et al., "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures," IEEE Computer, pp. 39-51, July 1990.

[4] Floyd, R.W., "Assigning Meaning to Programs," Proceedings Symposia in Applied Mathematics, Vol. 19, pp. 19-32, American Mathematics Society, Providence, Road Island, 1967.

[5] C.A.R. Hoare, "An Axiomatic Basis of Computer Programming," Comm. ACM, Vol. 12, No. 10, pp. 576-580, October 1969.

[6] L.G. Stucki, G.L. Foshee, "New Assertion Concepts for Self Metric Software Validation," Proceedings of International Conference on Reliable Software, pp. 59-71, Los Angeles, California, April 21-23, 1975.

[7] D.M. Andrews and J.P. Benson, "An Automated Program Testing Methodology and its Implementation," Proceedings of 5th International Conference on Software Engineering, pp. 254-261, San Diego, California, March 9-12, 1981.

[8] Aamer Mahmood, Dorothy Andrews and E.J. McCluskey, "Writing Executable Assertions to Test Flight Software," IEEE Trans. on Soft. Eng., pp. 262-266, 1985.

[9] D. Andrews, "Using Executable Assertions for Testing and Fault Tolerance," 9th Symposium on Fault Tolerant Computing, pp. 102-105, 1979.

[10] R.N. Taylor, "Assertions in Programming Languages," ACM SIGPLAN Notices, Vol. 15, No. 1, pp. 105-114, January 1980.

SOFTWARE RE-ENGINEERING:
A SYSTEMS AND OPERATIONS APPROACH

James LiVigni *
Capt. Richard Jernejcic USAF

EG&G Management Systems, Inc. *
RATSCAT Operations
Holloman AFB, NM 88330-7715

Abstract

One definition of Software Re-engineering is the analysis and redesign of an existing software system in order to create new software and associated processes that will improve the productivity, efficiency, and/or quality of the system or systems it replaces. If an effort requires not only re-engineering, but also a significant paradigm shift in operations due to limitations of the existing systems hardware and software, there is a significant risk of delivering a system that will actually decrease efficiency, quality and productivity or won't be used at all.

In order to lower the risk associated with this change, a number of methods can be utilized to obtain user inputs and more importantly ownership. These methods can be readily integrated into the formal systems engineering process. This paper will discuss the techniques used to obtaining user involvement in the re-engineering of the Data Acquisition and Processing Improvement Systems (DAPS) Embedded Resource System Software. Specific methods used in each portion of the development cycle to obtain user inputs and ownership will be discussed. Some observations about the user communities reactions to the effort and some conclusions as to the effectiveness of this approach are included.

I. INTRODUCTION

A. RATSCAT

The Radar Target Scatter (RATSCAT) Facility is the primary DOD facility for radar cross section (RCS) measurements. RATSCAT is comprised of two physically separate sites, Mainsite and RATSCAT Advanced Measurement System (RAMS) site. At RATSCAT, full-scale, flyable aircraft and missiles, operational vehicles, aerospace models, and miscellaneous targets can be accurately measured for radar target signature, both monostatic and bistatic, antenna gain and radiation patterns, and performance of active electronic systems. These measurements support weapons systems development, technology assessments, and related Department of Defense and U.S. Government sponsored efforts.

B. History

This paper is largely based on experiences from the Data Acquisition and Processing Improvement Systems (DAPS) project, an improvement and modernization program at the Radar Target Scatter Division (RATSCAT), Holloman Air Force Base, NM. The project was started in 1992 and initial operating capability (IOC) for the first system was April of 1995. The effort was to replace the data acquisition, control and processing subsystems from five separate Radar Measurement Systems, all having essentially the same mission but each having very different hardware, software, and operational implementations. The replacement systems was to have common processors and peripherals, as well as a common hardware and software architecture. The goal of this effort was to leverage training and software improvements across all sites, streamline operations, simplify hardware and software maintenance and provide for future growth. The DAPS effort would in effect change the operations paradigm for each specific Radar Measurement System, and provide a site-wide paradigm for control, collection and processing of RCS data. The fundamental philosophy of the effort was "operations should dictate the hardware and software....not the hardware and software dictate operations".

The system requirements were allocated to two subsystems. The first subsystem, called the Test Data System (TDS), would provide common test planning, test conduct, data management security and product creation. The second subsystem, called the Embedded Resource System (ERS), would essentially provide a common command, control and data interface regardless of the specific radar measurement system. For each sub-system a significant amount of re-engineering was required in order to subsume all the various existing systems functions and performance into a single suite of hardware and software.

C. Scope

This paper will discuss the techniques used to decompose the requirements , functions, performance, and operations of five separate Radar Measurement Systems into a single coherent requirements set. Methods used to obtain user inputs into the requirements analysis, design, fabrication, and test, as well as user ownership and support of the re-engineering effort, will also be discussed. A key element of these methods, specifically the use of the user interface as a focus for defining operational requirements, will also be discussed.

II. REQUIREMENTS ANALYSIS

A. Overview

Once the determination was made that the re-engineering effort would indeed improve quality, efficiency, maintenance, and reliability, the requirements analysis portion of the effort began. It was shown that the costs of the effort would be recouped by savings after deployment, and by the migration of the software to higher powered, lower cost systems. The systems analysis had three major areas of focus: (1) documentation leveling, (2) operational analysis, and (3) functional and performance analysis.

B. Documentation Leveling

During the requirements analysis phase of the effort, it was quickly determined that the existing documentation had portions that were either non-existent, incomplete, or incorrect. A single coherent set of existing software documentation would have to be developed and verified as correct before the analysis effort could begin in earnest. Since the existing software was located on different types of processors, each using different types of operating systems, it was felt that third party reverse engineering tools would not be cost effective or timely. Instead, some simple system description techniques for defining the software, calling structures, and interfaces was created. Figure 1 gives an example of the type of software system description diagrams that were generated. These diagrams served as the baseline for user interaction to determine the current versions of the software, their performance and functional requirements, and their interfaces. Once completed, the existing software engineering and configuration management personnel reviewed the documents for correctness.

There were two additional benefits to this effort that deserve mention. First, putting this baseline under configuration management served as a tool to provide feedback to the re-engineering team if there were any changes to the baseline during the development effort. Second, RATSCAT now a had a single coherent set of documentation.

C. Operations Analysis

Since the DAPS re-engineering effort was for subsystems integral to radar cross section (RCS) testing at RATSCAT, another portion of the analysis dealt with the role of the systems with respect to operations. What portions of the testing process were the software involved in, and how were the existing systems hardware and software limitations affecting the operational procedures? This involved breaking down the overall mission flow for analysis, and then providing a concurrency diagram to determine embedded systems resource requirements, based on operational requirements. Although the mission flow is shown as a sequential series of events, there is a certain amount of overlap and feedback which needed to be taken into account in determining system requirements. Although the DAPS mission covers Test Planning through Test Reporting, the embedded system requirements are essentially limited to the Test Conduct portion of the flow. Figure 2 shows the RATSCAT mission flow, with the test conduct portion expanded.

Based on analysis of test conduct for each site, a single system concurrency diagram was generated for the embedded system. This concurrency diagram provided the basis for analysis of common resource and processing requirements for each system. Figure 3 shows a system concurrency diagram highlighting major embedded system functions and temporal sequence.

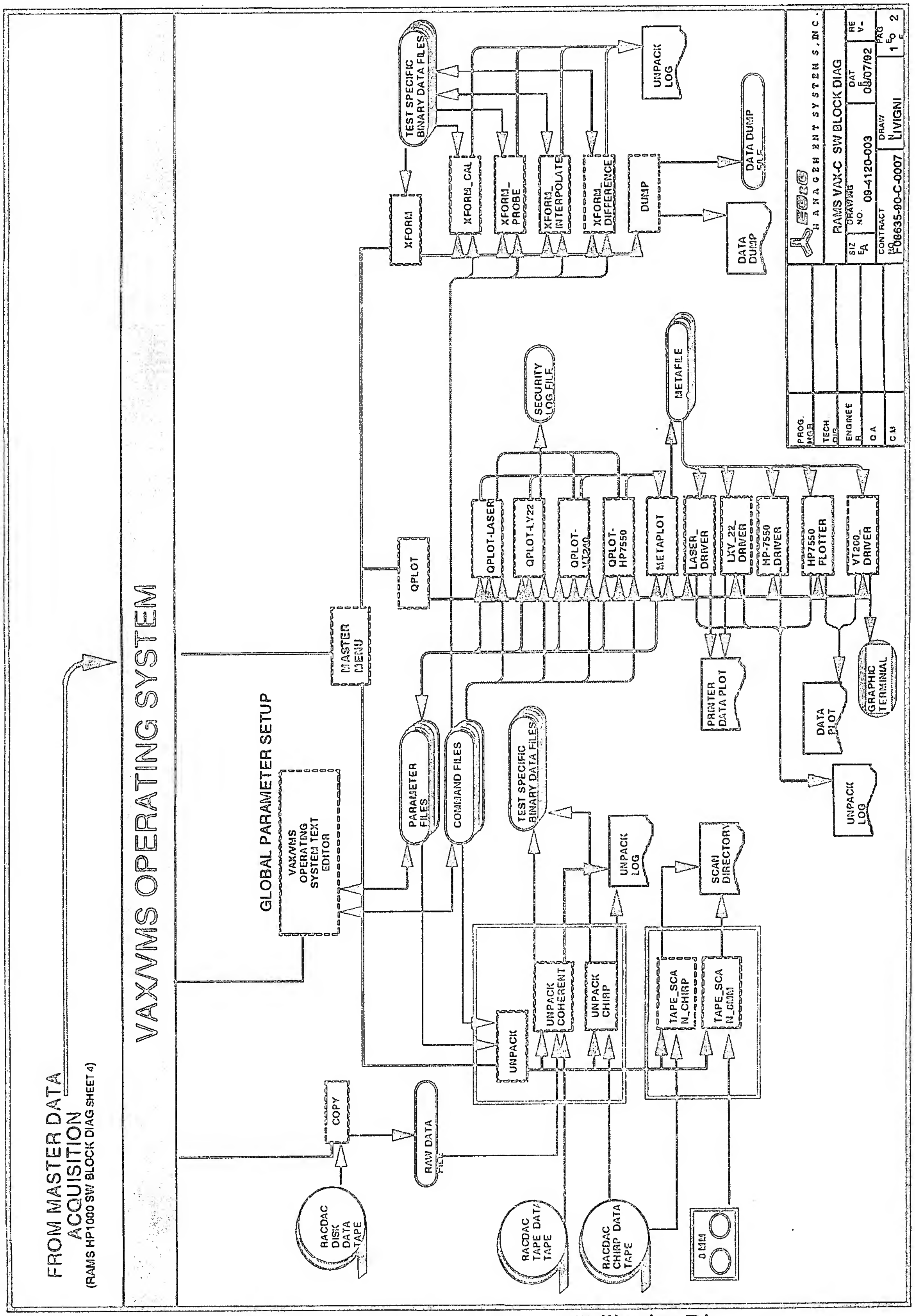


Figure 1 Software Structure And Resource Utilization Diagram

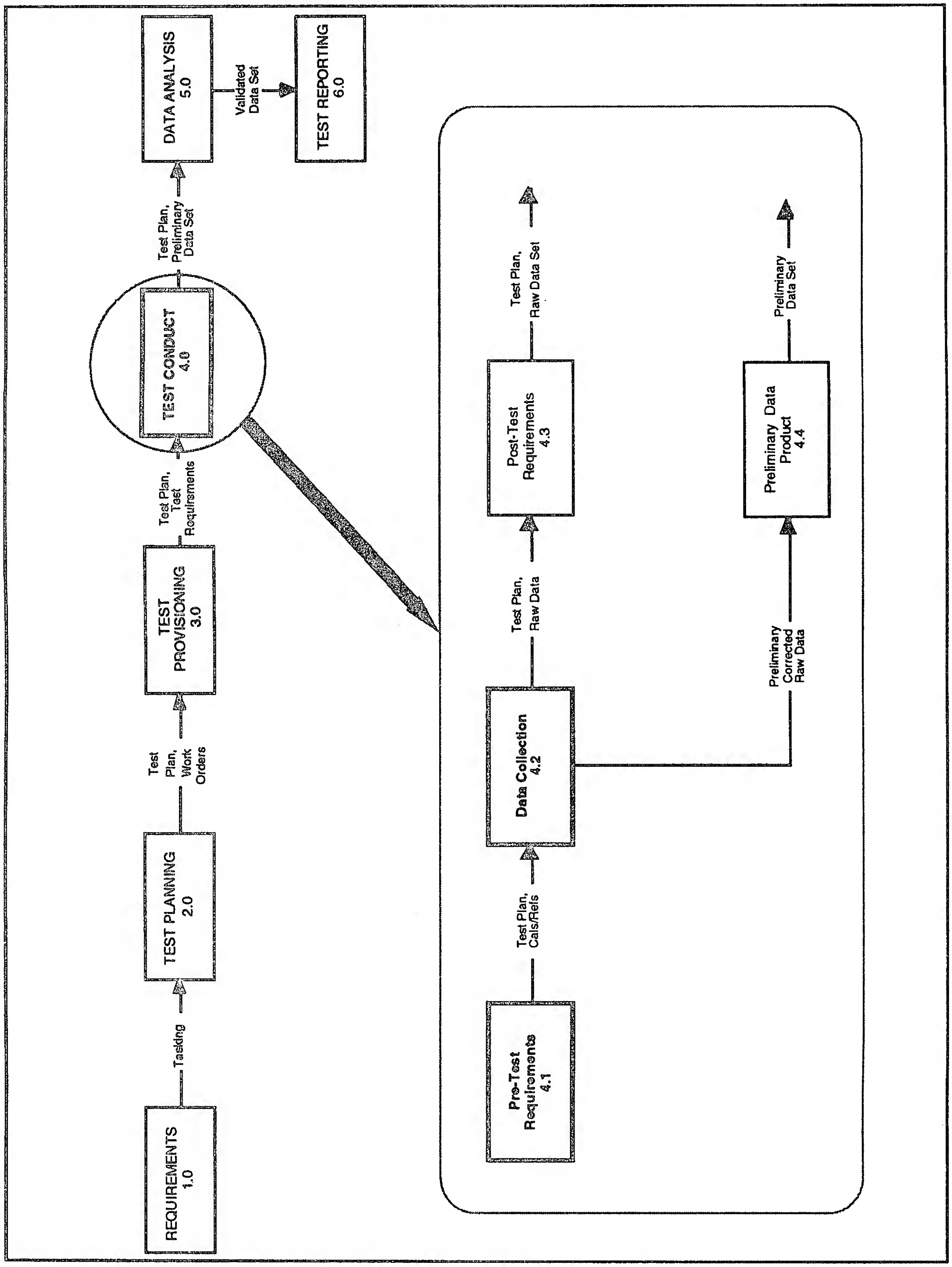


Figure 2 RATSCAT Mission Flow

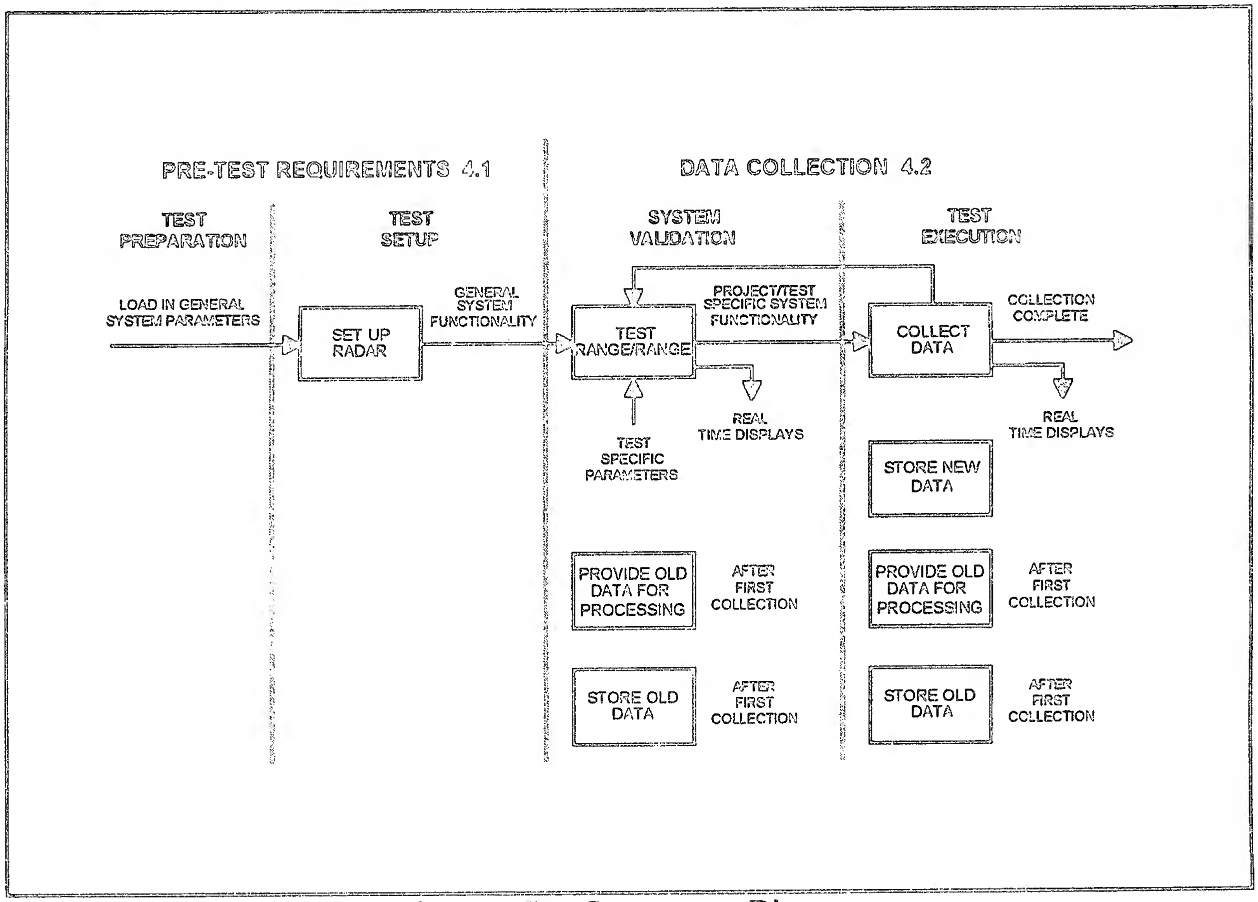


Figure 3 Test Concurrency Diagram

At this point, the user community became directly involved with the re-engineering effort through a number of activities sponsored by the re-engineering team. The first activity dealt with developing a standard set of nomenclature for the systems' parameters and embedded processing functions. The mission flow and concurrency diagrams were used as a guide for the discussions, and the requirements for each step for each site were reviewed. Based on this review, the re-engineering team in conjunction with the operations community came up with a site-independent set of procedures, processing, parameter organization, and nomenclature.

Given a consistent set of top level requirements, the next activity was to interview the operations staff. The interviews covered embedded system performance and utilization of each software package, redundant areas of processing and parameter entry, operational bottlenecks, etc. Another area of discussion dealt with desired system checks and status. The user community was also questioned in great detail as to the grouping , frequency, and organization of data for testing at each of the measurement systems.

D. Functional and Performance Analysis

Once the software engineering and operational inputs to the existing system were collected, the re-engineering team began defining the functions and performance of the new system. At this point, the re-engineering team had a good idea of what performance enhancements would result in improved operations. Also considered, was whether or not these could best be accomplished by

migration to another more powerful platform, by redesigning inefficient code, or a combination of both. This early involvement by the users set the context for the effort at the system level.

The overriding methodology for the analysis was object oriented; this aided in separating out *what* needed to be done from *how* to achieve it. It was found that this paradigm shift, while at first difficult to establish with the user community, allowed the team to more fully understand the operational requirements. It also allowed the users of different measurement systems to converse more freely with one another.

E. Requirements Synthesis

The final portion of the analysis involved the synthesis of a single set of requirements for the new software suite, based on the existing software functional and performance requirements, as well as operational requirements. Although the analysis was fundamentally object oriented, a method utilizing Software MIL Standard 2167a was established for requirements allocation and design documentation. The allocation of requirements to computer software configuration items (CSCIs), computer software units (CSUs), etc., and whether or not there was a one- to-one mapping of existing modules to new modules, was a function of many variables. Not the least of these variables were the new hardware architecture, new software standards, and available commercial off the shelf (COTS) software. The DAPS effort tried to utilize COTS where ever possible to reduce development costs and schedule, as well as to improve overall maintenance. Knowing what COTS was available to perform similar functions as those required by the new system, and early identification of them, aided immeasurably in the design of the system. Specifically, this knowledge minimized system interfaces and simplified the system architecture.

F. User Interface Development

Concurrent with the normal development cycle, and integral to the rapid prototyping effort, were the user demonstration. The motivation behind these demonstrations was to utilize the prototype user interface as a focus for discussions with the operations community. The user interface provided a framework within which the specific issues of parameter management, diagnostics, operational procedures, etc., could be reviewed. In reality, there were a number of user demonstrations throughout the effort; however, they can be grouped into three major demonstrations. These were as follows: (1) Navigation and Representation; Mechanisms and Capability, (2) DAPS System Navigation/Operational Concepts, and (3) Pre-release. These three demonstrations occurred during the Analysis phase, System Design phase, and Fabrication and Test phase, respectively.

G. User Interface Demo 1: Navigation and Representation; Mechanisms and Capability

The first user interface demonstration occurred during the analysis phase of the project. The goals of this demonstration were twofold. One, familiarize the user community with the graphical representations that will be used to develop the interface. Two, begin to get user inputs as to applicability of specific items to operational requirements (parameter entry, visualization, feedback, etc.). The demonstration was restricted to a review of each major type of widget and possible attributes. As part of the review, the users were provided screen dumps and demonstra-

tions of similar user interfaces used in industry. At the conclusion of the review, the users were polled as to their preferences for parameter entry, visualization, and feedback mechanisms.

As a result of this first demonstration, the re-engineering team obtained some fundamental information as to what the operations personnel felt was optimal for the type of testing that occurs at RATSCAT. For example, the team learned that parameter entry was a chief source of error; if choices of parameter values could be provided via pull down menus, it would increase efficiency and reduce or eliminate a source of error. In addition, it was found that for most testing needs, graphical feedback of control parameters was not required. However, graphical real time displays must be available and configurable in size and content. This information, gleaned early on, afforded the interface developer insight that could not be fully obtained from the existing systems. Those systems were *very* limited with respect to user interfaces.

III. SYSTEM DESIGN

A. Overview

During the design process, a number of design issues came up that would impact operations. In each case, these impacts were discussed with the users and a decision was reached based on both programmatic and operational requirements. For example, as the overall hardware and software architecture evolved, it was observed that the design would be simplified if the data did not have to be processed as acquired (i.e. in bursts), but was instead buffered and processed as a continuous stream. The operational impact of this buffering was discussed with operations personnel and the design approved. Because the design team requested inputs from operations personnel and provided feedback as the design progressed, the user community felt that they were indeed part of the design effort, and that the system was not being developed in a vacuum.

B. User Interface Demo 2: DAPS System Navigation/Operational Concepts

During the system design, the second major user interface demonstration was held. This demonstration basically covered the skeleton of how the system would be accessed and operated by the users. It provided valuable feedback on inputting of parameters, organization of data, type of visual feedback required, and the concept of operations of the new system. This feedback, received early in the design process, required much less rework later in the integration and test cycle. In addition, the user community had inputs into, and review of, the system design and its resulting concept of operations.

A specific example from the DAPS effort involved the requirement for access control and auditing. Quality assurance wanted the system to provide access control and accountability for parameter changes made during testing, so as to minimize error and aid repeatability. The users on the other hand contended that this would prevent logging out and in during testing, such as for overlapping shifts and lunch breaks. A compromise solution, labeled 'shift change,' was developed based on these discussions. 'Shift change' would allow one user to log off and another to log on while a test was occurring.

IV. FABRICATION AND TEST

A. Overview

During the fabrication and test phase, the user community provided inputs to the development of the acceptance test. As part of the overall DAPS acceptance test effort, a worksheet was developed to describe what was required by way of an acceptance test. This worksheet was used by the user community to provide information as to the number and types of tests they deemed necessary. After the users submitted their inputs, the re-engineering staff reviewed and discussed any modifications and additions they felt were appropriate, before submitting both to the RATSCAT technical director for final review. Figure 4 shows a completed worksheet. As a validation of the test development effort, a select group of operations staff were invited to actually perform portions of the acceptance tests.

RAMS DAPS ACCEPTANCE TEST PLAN QUESTIONNAIRE		
Test Name	Attenuator Checks	
Objective	To analyze the difference in amplitude levels with respect to the difference in the attenuator setting for each given data cut	
Description of Test	Perform pseudo azimuth data cuts on the target (cylinder) while making adjustments to the RF and IF attenuation's for both HH and VV polarization parameter sets. Parameter sets associated with the small antennas only will be used in this test. Measurements will be conducted on an uncalibrated system. The PRF will also be varied to check its effect on the attenuation levels.	
Test Radar/Pylon Configuration Required	Radar Parameters The radar is to be configured according to the narrowband test parameters identified in the table below. Menu sets must be built to accommodate these radar parameter requirements.	
	FREQUENCY SETS	1 through 7
	POLARIZATION	HH & VV
	PRF (kHz)	40, 20,10
	PULSEWIDTH (ns)	185
	RSS (ns)	85
	VIDEO BANDWIDTH (MHz)	50
	IF ATTENUATION (dB)	(0-30)/1 dB Steps, (30-90)/10 dB Steps
	RF ATTENUATION (dB)	(0-45)/3 dB Steps
	INTEGRATION	512
Pylon Hardware Required	As per Table 1.2-1	
Test Procedure	Using frequency set 5, check each IF attenuation listed in the table above. PRFs will also be checked by using only the frequency set 5. Perform measurements while changing one parameter at a time. Example: RF = 0, PRF = 40, change IF from 0 through 30, 1 dB steps ands 30 to 90 at 10 dB steps. Testing of the RF attenuator controls must be performed for each band by selecting menu sets that will accommodate this. Example: IF = 0, PRF = 40, change RF from 0 through 45 in 3 dB steps.	
Criteria for Completion	Compare to current MKIII results within ± 0.5 dB	

Figure 4 Completed Acceptance Test Worksheet

B. *User Interface Demo 3: Pre-Release*

This demonstration basically covered the entire user interface prior to acceptance testing, in order to insure that the user interface had the proper functionality to perform each aspect of a test program. The users were asked to actually dry run the ATP, and any problems that precluded successful testing were resolved.

V. OBSERVATIONS

Any effort that results in change is bound to meet with resistance, and re-engineering is no exception. The basic premise, that this change is going to make life easier for the user, is generally disbelieved at the outset for a number of reasons. Not the least of these is a lack of understanding and communication between the users and the re-engineering team. The DAPS project as a whole was subject to this resistance, perpetuated by both distant past and more recent experiences with hardware and software deliveries at RATSCAT. On the whole, as the project progressed this resistance gradually faded. This was due in no small part to the users' gradual recognition that their inputs *were* reflected in the system design reviews. The following represents the general evolution of the users' reactions to the attitudes as the development progressed.

- *Guarded mistrust of the entire concept*
- *General speculation that operations is now doomed*
- *Unbelief that inputs will be taken into consideration*
- *Growing realization that system does reflect their inputs*
- *Enthusiastic user participation and debate*
- *Core group of highly participative users emerge*
- *Anticipation by users of new system*

This shift in the users' perceptions resulted in the following benefits.

- *Core group acts as liaison to community simplifying feedback*
- *Core group participates in ATP*
- *Minimal software changes and high initial utilization*
- *Rapid integration into operations*

VI. CONCLUSIONS

A. *Feedback Sources*

In general, it was found that the software engineering staff had the best insight into resource utilization of the existing software, as well as the performance of specific routines and packages. The user community, on the other hand, provided a wealth of details on how long it took to perform specific test operations utilizing the existing tool suite, as well as recurring sources of error.

B. *Benefits*

Overall this "use the users" approach was successful in reducing users' resistance to what were in some cases fundamental changes in day to day operations. The re-engineering team came to be recognized as a group of people who really wanted the users to be successful in their jobs. One of the largest benefits from the re-engineering effort was it focused the users of each individual measurement system on the more global concept of RATSCAT as a single operational entity. It exposed users to the advantages and disadvantages of each measurement system. Furthermore, it fostered the idea that the design tradeoffs made, while not necessarily ideal for any one system, were the best for RATSCAT as a whole.

C. *Insuring Future Benefits*

In order to insure the continued benefits from the delivered systems, care must be taken to insure that individual systems do not de-evolve into separate operational entities again. At RATSCAT, a semiformal configuration management team is being established to insure that new parameter names, file names, product types, and operational procedure changes will have site-wide review prior to site-wide implementation.

A CHANGE MANAGEMENT TOOL FOR AVIONICS SOFTWARE MAINTENANCE

Dr. Joseph P. Loyall*
Susan A. Mathisen

TASC, Reading, Massachusetts 01867-3297

James S. Williamson

Wright Laboratory, WPAFB, Ohio 45433-6543

Abstract

TASC, under contract to Wright Laboratory,[1] *has been performing research into advanced automated verification and validation (V&V) techniques for avionics software. As part of these efforts, we developed a comprehensive V&V process for the software life cycle, examined the current software maintenance process at several USAF Air Logistics Centers (ALCs), identified potential areas for automated support of this process, and advanced the theory and practice of V&V techniques. We are currently developing a change management tool that provides automated support for avionics software maintenance and is based upon our V&V process and our advances in dependence analysis. The tool identifies the effects of software modifications, assisting a software maintainer in evaluating the suitability of a proposed software modification, selecting between several possible modifications, measuring the complexity of a program before and after a software modification, driving regression testing, and selecting unit test cases. This paper describes the motivation behind the change management tool including our V&V process and the avionics software maintainer surveys, the architecture of the tool, our current and future development, and our plans for future research and development.*

1. Introduction

The maintenance of Air Force legacy systems is becoming increasingly important as avionics software systems become larger and more complex. Yet the maintenance of these software systems becomes increasingly difficult over time because of the following:

- As many modifications are made to a software system, it often becomes larger and more complex, and diverges from its original requirements and design documentation.

- Because of employee turnover, maintainers with many different styles and varying levels of understanding of the software system make modifications, contributing to the increasing complexity of the system.

- Since each upgrade often contributes new test cases to a software system's Acceptance Test Procedure (ATP) and tests are rarely, if ever, removed from the ATP, it becomes increasingly difficult to determine when there is redundant or incomplete testing of a software system. Executing the complete ATP becomes increasingly time-consuming and tedious, resulting in fewer modifications per block upgrade.

[1]This work is being performed for the USAF Wright Laboratory under the Advanced Avionics Verification and Validation Contract (Contract No. F33615-91-C-1704) and Advanced Avionics Verification and Validation Phase II, Delivery Order No. 5, Avionics Software Technology Support Program (Contract No. F33615-92-D-1052/0005).

- Implementation of modern aircraft features with new software systems and modifications to existing software systems involves increasing complexity, which causes the training of new avionics software maintainers to become more expensive.

- Finally, unlike hardware maintenance costs which scale with the number of operational aircraft, software maintenance is a fixed cost per weapon system that has not seen cost savings due to Air Force drawdowns.

Other factors, such as employee turnover and division of modifications between maintainers, can contribute to the difficulty in understanding and maintaining avionics software over time. A *change management* system, which aids in evaluating and performing maintenance without increasing the complexity of the software system is important to help minimize the expense and effort of avionics software maintenance, increase the modifications possible per block upgrade, and to detect problems early in the software maintenance and testing process.

TASC and Wright Laboratory have been performing research into advanced automated Verification and Validation (V&V) techniques for the Advanced Avionics Verification and Validation (AAV&V) and the AAV&V Phase II programs. The goals of the AAV&V programs are to advance the state-of-the-art in automated V&V techniques and tools for real-time, mission-critical software, including testing, error detection, and software reliability measurement, as well as improvements in software understanding and maintenance. The AAV&V programs include design of an AAV&V process [5], development of a proof-of-concept system illustrating automated support for portions of the process [6], and development of improvements in V&V techniques such as dependence analysis [7].

As part of the first phase of the AAV&V program, we designed a comprehensive V&V process for avionics software, conducted a survey of avionics software developers and maintainers at three different Air Logistics Centers (ALCs) to understand the current avionics software V&V and maintenance process, identified the portions of the AAV&V process that could most benefit avionics software maintainers, and developed new theory and techniques for utilizing program-wide dependence analysis and coverage analysis in software V&V. As part of the second phase of the AAV&V program, we are currently developing a change management tool based upon the results of the first phase. The change management tool supports the avionics software maintenance process that we observed in the following ways:

- The tool uses the latest advancements in dependence analysis (including those developed during the AAV&V program) to implement a *change management* capability for the maintenance of avionics software. The tool indicates the ways in which parts of a software system can influence the behavior of other parts of the system. This improves software understanding, since dependence analysis indicates the control flow and data flow characteristics of a program and provides powerful metrics indicating modularity, complexity, coupling, and cohesion characteristics of the program. Dependence analysis also helps manage the increasing complexity of software because it indicates the potential impact of a software modification on other parts of the program. This information enables a software maintainer to evaluate whether a potential software modification is appropriate, whether it has unexpected potential side effects, and whether it negatively impacts the complexity of the program. Once it has been decided that a software modification is necessary and appropriate, dependence analysis can indicate the parts of the program that need to be regression tested, aid in the selection of tests for regression testing, and indicate the need for additional test data.

- The tool includes a graphical user interface that includes hyperlinked graphical program views that utilize color and icons to convey information. The graphical views reduce the learning curve that must be overcome by new software maintainers or by experienced software maintainers updating unfamiliar sections of code. The hyperlinked graphical views aid the maintainer in filtering and navigating information. For exam-

ple, a maintainer can click on an icon that indicates a potential effect of a software modification to open up a source code window with the affected code highlighted.

- The tool currently supports avionics software written in Ada and a later version will support software written in JOVIAL. This ensures that the tool supports both a significant portion of legacy systems currently being maintained and future avionics systems that will need to be maintained.

This paper describes the results of our AAV&V research, including the AAV&V process and ALC surveys, and our ongoing AAV&V tool development. The paper is structured as follows: Section 2 describes the AAV&V process we developed during the first phase of the program. Section 3 presents an overview of the current avionics V&V and maintenance process, including possibilities for automated support and difficulties in automating the process. Section 4 describes the AAV&V change management tool. Section 5 presents our plans for future research and development. Finally, Section 6 presents some concluding remarks.

2. Overview of the AAV&V Process

As part of the first AAV&V program, we examined the state-of-the-art in V&V techniques and tools and defined a process for V&V throughout the software life cycle of real-time, mission-critical avionics software, including requirements verification, error detection, testing, verification of critical functionality, and estimation of software reliability. The AAV&V process, described in detail in [5], supports the entry of requirements specifications, design specifications, and information needed to test and document the software. It also supports the V&V of existing software.

When possible, requirements and design information should be captured and used to aid in software understanding, analysis, and the selection of test cases for V&V. In many cases, however, such information is absent, out-of-date, or incomplete, leaving only the software's source code as a reliable input to a V&V process. The AAV&V process therefore includes both the capacity to capture requirements and design information early in the software life cycle and to accept only source code and record software requirements and design information as it is uncovered during maintenance.

Many current V&V processes or tools rely on one V&V technique, such as functional testing or code inspection. However, as software becomes more complex and entrenched in critical systems where failure could lead to catastrophic results, it is important to rely on a variety of complementary techniques. The AAV&V process includes the following classes of V&V techniques:

- *Static Analysis* V&V techniques examine the static structure of source code to detect errors and verify program properties. Static techniques are important because they are amenable to automation and they are useful for detecting errors quicker and easier than with other techniques. Static techniques alone are often not sufficient because they detect only certain types of errors, e.g., syntax errors and run-time errors caused by static characteristics of the code, and only approximate other errors and program characteristics, e.g., indications that deadlock *might* occur in a program.

- *Structured Testing* techniques execute (or simulate execution of) a program with carefully selected test data to detect errors or verify properties. Structured testing can detect run-time errors that static analysis cannot and it can sometimes indicate when a program does not satisfy its requirements. However, structured testing is usually more effort-intensive than static analysis, can only approximate adherence to requirements, and is only as good as the test selection technique used.

- *Formal Analysis* specifies the requirements of a program in a formal, mathematical notation and validates the program's adherence to its requirements using mathematical deduction. Formal analysis is important for critical systems to formally validate satisfac-

tion of requirements. However, it can be prohibitively expensive for anything but small code sections and is not yet supported well by automated tools.

- *Statistical Analysis* performs testing of a program using a randomly chosen input data set. Statistical analysis is useful for providing a statistical estimate of software reliability because it removes tester bias from test case selection and adds formalism to the testing process. However, it often requires a large number of test cases to achieve reasonable reliability estimates. Even though its test case selection can be automated, it often requires more effort to verify testing results.

In the AAV&V process, static analysis is used first to quickly and automatically detect errors and potential requirement violations. Then structured testing is used to detect additional errors, test particular requirements, and gain confidence that requirements are satisfied. Formal analysis is used to validate critical requirements and statistical analysis is used to estimate reliability and minimize the chance of errors being undetected due to biased test set selection.

The AAV&V process is flexible, however, and allows the order and level of V&V effort to be tailored for particular applications or domains. For example, non-mission critical software can be verified using only a few static analysis and structured analysis techniques; the consequences of failure do not justify the effort required for formal and statistical analysis. As another example, the early stages of critical software development might emphasize formal analysis to ensure that design specifications fulfill the program's requirements; structured testing is not necessary until implementation of the design.

As part of the first phase of the AAV&V program, we developed a proof-of-concept system that demonstrates static analysis and structured testing techniques and their applicability to avionics software maintenance and V&V. We are currently developing an AAV&V tool that implements several static analysis techniques. We plan to interface this tool with existing structured testing tools and to explore implementation of formal analysis in future work.

3. Software Maintainer Surveys

We conducted a survey of the maintainers of F-15, F-16 and E-3 Operational Flight Programs (OFPs) to identify the current avionics software maintenance process and where automated support would be most useful. This section describes the maintenance process observed during our surveys and the challenges faced by developers of automated tools for the maintenance process.

3.1 The ALC Software Maintenance Process

Although the avionics software maintenance process varies for each aircraft, the following general maintenance process, illustrated in Figure 1, was observed at each ALC [11]. Typically OFPs are developed by contractors and then turned over to ALCs for organic maintenance. The maintenance process relies heavily on the OFP maintainer's expertise and knowledge of the code and test procedures. A change request to fix a bug or add new functionality in the aircraft is forwarded to the OFP maintainer. The OFP maintainer reviews the change request and determines the modules of the OFP that must be modified to implement the change. The maintainer modifies copies of those modules and performs unit testing of the changed modules. A new OFP is created by incorporating all changed modules into the OFP and the complete OFP is tested by executing its ATP, a set of test procedures that exercise functions of the OFP. The ATP continues to grow over time as new tests are added for change requests.

Our survey of OFP maintainers indicated the desire for improved automated support for OFP maintenance, including the following:

- *Determining the possible effects of OFP modifications.* Currently the software maintainer relies upon his knowledge of the OFP code and his experience to estimate the impact of his modifications on the rest of the program and to determine the portions of the

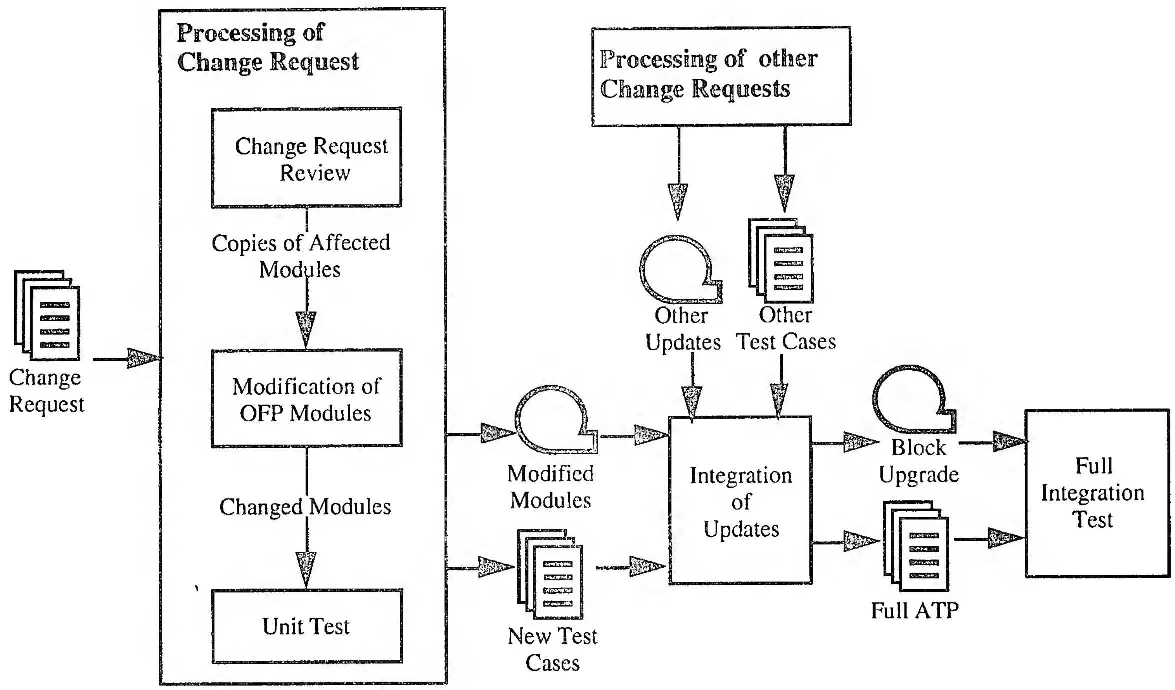


Figure 1: The Avionics Software Maintenance Process

OFP to regression test. As OFPs are modified and become more complex over time, and as new maintainers are introduced, there is an increased possibility that errors introduced during modification will go undetected until integration testing.

- *Selecting and developing test cases.* Avionics software maintainers are faced with an ever-increasing ATP from which to select test cases for unit testing. Executing the complete ATP is effort-intensive, and is therefore reserved for integration testing before software is fielded. The software maintainer relies upon his knowledge of the code and experience to select test cases and, when necessary, to develop new test cases. If his selection is insufficient (which becomes more probable as OFPs become more complex and after many modifications, and when a maintainer is inexperienced), the probability of errors going undetected until integration test is increased.

- *Executing tests and verifying results.* Unit testing and integration testing are usually performed manually and are tedious and time-consuming to perform. For example, the F-15 Central Computer OFP ATP currently requires two person weeks to perform. Much of this effort is manual setting of switches and verification of displays. As OFPs get larger and more complex and the ATPs continue to grow, manual execution of ATPs will become impractical.

- *Determining test coverage.* Currently, maintainers have to rely on software inspections to estimate how thoroughly their selected test cases exercise the OFP code. The possibility of insufficient testing increases when the code is unfamiliar and complex or the maintainer is inexperienced.

3.2 Potential Automated Support for the Software Maintenance Process

Several existing V&V techniques can be used to reduce the effort involved in maintaining avionics software. Automated *dependence analysis* [9] traces the interactions between parts of a program and indicates the impact that modifications of a section of code can have on the rest of a

program. Dependence analysis can detect potential errors introduced by modifications early enough to prevent them, unit test for them, and correct them before costly integration testing.

Coverage analysis [1] automatically traces the parts of a program exercised by test cases during testing. Coverage analysis can indicate insufficient test sets and unexpected program characteristics, e.g., dependences or paths introduced by modifications, during unit testing, reducing the expense and trouble associated with detecting them during integration testing or after the software is fielded. As OFPs get larger and more complex, it becomes more difficult to manually determine whether an OFP is tested thoroughly. Automated coverage analysis is necessary to determine whether any parts of the OFP are missed during testing and the extent to which the tested parts are covered.

Dependence analysis and coverage analysis can be combined to provide automated support for test case selection and generation and to reduce the amount of testing necessary. The parts of an OFP affected by a modification can be automatically determined using dependence analysis. Coverage data from previous unit tests can be accessed to automatically select test cases that exercise the affected parts. Other parts of the ATP would not need to be executed during unit testing because the portions of the OFP they exercise are unaffected by the modification.

In addition, *automated test execution* and *result verification* alleviate the tedium involved with performing unit and integration testing. Software maintainers can execute test cases as a batch process and examine the results at their leisure.

3.3 Difficulties in Automating the ALC Software Maintenance Process

We observed several difficulties with automating the avionics software maintenance process. The following paragraphs describe these and discuss some possible solutions.

Avionics systems utilize a variety of languages and platforms. Currently avionics software is written in a variety of languages, including JOVIAL, FORTRAN, Ada, and many variations of Assembly. Even within a single aircraft, OFPs are often written in several different languages. Many of these languages, such as JOVIAL and special-purpose assembly languages, are not supported by commercial software tool developers. The only automated tools are tools developed specifically for the single ALC customer, either in-house or by contractors. In addition, OFP systems are often hosted on special-purpose processors with their own operating systems. Any OFP software targeted toward special-purpose hardware requires specially developed compilers and support systems, such as debuggers. Because of the expense and effort involved with their development, many ALC maintainers do not have access to symbolic debuggers or other automated support software. The move toward common languages, such as Ada, and commercial hardware systems promises to improve the situation, but legacy systems will be around for some time.

Since the maintainers are not usually the original software developers, there is a large learning curve to understand the requirements and design information. Usually, OFP software is developed by a contractor and the finished system is turned over to the ALC for maintenance. Even though requirements and design documents are usually included in the delivery, many design decisions might be lost in the transition. At the beginning of organic maintenance of an OFP, software maintainers encounter a large learning curve to become familiar with the software and much of the maintenance process is spent improving the maintainer's understanding of the code. As the maintainer becomes more familiar with the OFP, software modifications become quicker and introduce fewer errors. The OFP maintenance process relies heavily on the understanding and expertise of individual maintainers to select the portions of an OFP to modify, to perform the modifications, and to select tests for unit testing. Each time a personnel change occurs, the learning curve is encountered anew.

Standardized documentation (i.e., DOD-STD-2167A) helps alleviate the problem by capturing requirement and design information. Using a commercially-available set of software design environment tools, along with delivery of all documentation produced by these tools, will improve the situation further.

Avionics software is usually embedded in a larger system. Testing an embedded OFP or component often requires inputs from other embedded OFP components. Testing, therefore,

should occur in an environment which simulates, emulates, or includes embedded avionics equipment and software. Each of these components often have real-time, memory, or space constraints. This requires special-purpose test environments and limits the effectiveness of commercial test software. Some host-based V&V can be performed, e.g., to perform static analysis or to test the logic of a program in the absence of its timing and memory constraints. Ultimately, however, the OFP must be tested in a realistic embedded environment.

The USAF Advanced Multi-Purpose Support Environment (AMPSE) and COmmon Modular EnvironmenT (COMET) programs are currently building extensible, multi-purpose avionics test facilities [3]. AMPSE and COMET are based upon a modular design where simulators, emulators, and actual avionics components can be swapped in and out to model different aircraft configurations. These environments will reduce the expense and effort involved with supporting different OFPs and provide an environment with which AAV&V software tools can interact.

4. The AAV&V Change Management Tool

We are currently developing an automated software impact analysis tool to provide support for the avionics software maintenance process. The current tool concentrates on employing *dependence analysis* to analyze the ways in which portions of a program interact. The tool will assist avionics software maintainers in understanding software systems, modifying OFPs in such a way as to minimize system-wide impacts, and performing regression testing more efficiently. We plan to explore interfacing with automated test execution and coverage analysis tools in future versions to provide additional automated support for the maintenance process.

The AAV&V tool's dependence analysis provides the following capabilities to the avionics software maintainer:

- *Evaluate the appropriateness of a proposed software modification.* The software maintainer can use dependence analysis before a modification is performed to determine the portions of the program that might be affected. If a large number of disconnected components or critical portions of the program might be affected by the modification, the maintainer should consider an alternate solution.

- *Determine the portions of a program that need to be regression tested.* Once a modification has been performed, the software maintainer uses the tool to determine the components of the program that could be affected by the modification. All of these components must be regression tested. The tool also prevents the maintainer from performing unnecessary regression testing on components whose behavior couldn't possibly have changed because of the modification.

- *Improve software understanding and training.* Over time, the software maintainer will become more familiar with the control and data flow characteristics of the program from using the tool. This will enable him or her to perform quicker and better software maintenance.

The AAV&V tool's architecture is illustrated in Figure 2. It consists of the following components:

- *Graphical user interface* – The AAV&V tool provides an OSF/Motif[2] interface to invoke its functionality, such as analyzing a program, performing a dependence query, saving or retrieving information from the database, or displaying views of a program's structure. The tool provides several views, including a control flow graph, dependence graph, a diagram of the syntactic declarations in the program, and a source code listing.

---

[2]OSF/Motif is a trademark of the Open Software Foundation, Inc.

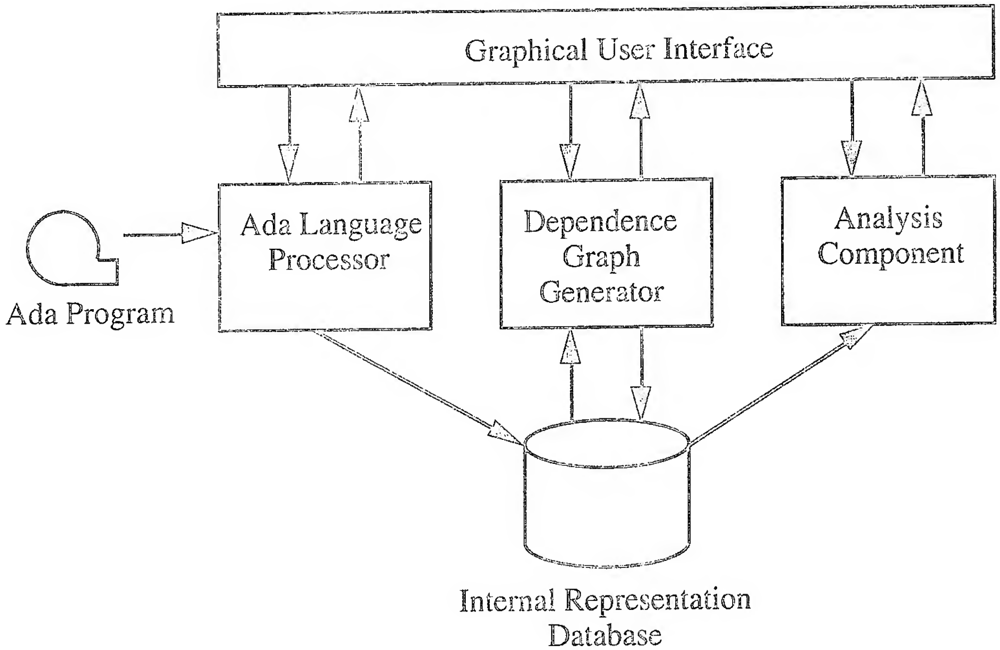


Figure 2: Architecture of the AAV&V-II Static Analysis tool

All of the views are *hyperlinked*, so that moving the mouse pointer over an item in one view highlights the corresponding portions of the other views. The interface supports browsing of the dependence and control flow graphs and querying the analysis component.

- *Ada language processor* – This component reads programs written in the Ada language and creates internal representations containing the program's syntactic and semantic information.

- *Dependence Graph Generator* – This component generates representations containing information about the control flow, data flow, and dependence characteristics of the program from the internal representations generated by the language processor [10].

- *Analysis Component* – This component examines the dependence graphs to extract information requested by the avionics software maintainer. It supports *what if* scenarios to help a maintainer understand a software system and evaluate the consequences of proposed software modifications. It also supports determining the dependence characteristics of a program before and after a modification, assisting in regression testing, evaluating a modification's impact on program complexity, and indicating whether additional test cases must be created. The results of a query are indicated by highlighting, shading, and color on the dependence graph. The hyperlinking mechanism can be used to determine the source code or declarations corresponding to an indicated dependence.

- *Internal Representation Database* – The database stores the internal representations used by components of the tool.

The AAV&V tool is currently under development. It is written in Ada and hosted on a SUN platform, using UNIX, the X Window System[3], and OSF/Motif.

[3]The X Window System is a trademark of the Massachusetts Institute of Technology.

5. Future Research and Development

This section describes some of our short-term and long-term plans for continuing research and development [8].

5.1 Support for Other Languages

Our choice to support the Ada language has helped gain support for our AAV&V work within the DoD community because of the DoD move toward Ada systems. However there are few Ada-based avionics systems that can benefit from an Ada-specific tool today. Consequently, we built our tool around Arcadia's language-independent IRIS (i.e., Internal Representation Including Semantics) representation [2, 4], so that minimum effort would be involved in additional language support. We can automatically leverage additional Arcadia advances and language front-ends, integrating them as they become available. In addition, we can develop front-ends for the prototype using widely-available parser generators and adapting them to generate IRIS representations. We have plans to develop a JOVIAL version of the tool under the current effort.

The techniques developed and exploited by the AAV&V system support future languages as well as existing languages. The IRIS and flow graph representations can be generated from any class of language, from procedural languages like Ada and JOVIAL to object-oriented languages such as Ada 9X. Dependence analysis is performed on information extracted from IRIS graphs and flow graphs and therefore is effective with different classes of languages. For example, data dependences in object-oriented languages center around the actions performed on objects while control dependences can exist because of method calls and inheritance.

5.2 Interfacing with Other USAF V&V Efforts

There are several ongoing DoD efforts to improve the testing of defense software, such as the USAF AMPSE and COMET programs. These programs are developing test environments, simulators, test execution engines, and result verifiers to support the ALC testing process. The Automated Validation (AutoVal) system [12] used in the AMPSE program provides a scripting language for describing long test procedures and an execution engine that automatically executes AutoVal test scripts and verifies results.

The AAV&V tool can be integrated with the AMPSE environment. The AAV&V tool would provide static analysis capabilities to increase software understanding, detect errors, aid in test case generation, and aid in coverage analysis. The AutoVal engine would provide test execution, automated result verification, and an interface with simulators.

5.3 Development of Complementary V&V Technology

The AAV&V process we developed includes several V&V techniques that complement the capabilities of dependence analysis and the AAV&V tool. For example, coverage analysis used in conjunction with dependence analysis can be used to automatically select test cases from an ATP. To accomplish this, coverage information is saved to a database and indexed by the components of the software system that are covered. When dependence analysis indicates the components of a software system that need to be regression tested, the database is accessed to extract test cases that have previously covered those components.

In addition, formal analysis, requirements capture, and software metrics are useful in conjunction with dependence analysis for early detection of errors, measuring adherence to requirements, and reliability estimation. We plan to continue our research in these areas and to incorporate results into the AAV&V tool.

6. Conclusions

We are currently developing a software maintenance tool that supports the maintenance of USAF avionics software. The AAV&V tool performs automated *change management* of a software system, which is useful in evaluating the suitability of a proposed software modification, selecting between several potential modifications, measuring the complexity of a program before and after a software modification, determining the components of a program that must be regression tested after a modification, and selecting test cases for regression testing. The tool provides a graphical user interface for performing *what if* queries and for analyzing and browsing a program's structure and semantics.

The development of the AAV&V change management tool is based upon research we have performed in avionics software V&V. It is based upon our AAV&V process, builds upon advances that we have made in V&V techniques, and supports the avionics software maintenance process we observed during our ALC surveys. Development of the AAV&V change management tool is part of a long-term strategy for researching and developing improved software V&V technology and tools for avionics software. We are continuing our research and development in the scope of our comprehensive software V&V process, with plans to investigate formal analysis and coverage analysis of avionics software.

References

[1] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, NY, 1990.

[2] K. Forester, "IRIS-Ada Reference Manual Version 1.0," Arcadia Technical Report UM-90-07, University of Massachusetts, Amherst, MA, May 20 1990.

[3] G. Howell and J. McCord, "Support Environment Methodologies for the Rapid Reprogramming of Operational Flight Programs," *Proceedings IEEE Systems Readiness Technology Conference*, San Antonio, Texas, 1990, pp. 275-280.

[4] R. Kadia, "Issues Encountered in Building a Flexible Software Development Environment," *Fifth Symposium on Software Development Environments*, McLean, Virginia, December 1992, pp. 169-180.

[5] J.P. Loyall, S.A. Mathisen, P.J. Hurley, J.S. Williamson, and L.A. Clarke, "An Advanced System for the Verification and Validation of Real-Time Avionics Software," *Proceedings 11th IEEE Digital Avionics Systems Conference*, Seattle, WA, October 1992, pp. 370-375.

[6] J.P. Loyall, S.A. Mathisen, P.J. Hurley, and J.S. Williamson, "Automated Maintenance of Avionics Software," *Proceedings IEEE NAECON*, Dayton, OH, May 1993, pp. 508-514.

[7] J.P. Loyall and S.A. Mathisen, "Using Dependence Analysis to Support the Software Maintenance Process," *Proceedings of the Conference on Software Maintenance*, Montreal, Canada, September 1993, pp. 282-291.

[8] J.P. Loyall and S.A. Mathisen, "Automated V&V Support for Avionics Testing," *Proceedings of the ITEA Test Technology Transfer Symposium*, Newport, RI, August 29-September 1 1994.

[9] A. Podgurski and L.A. Clarke, "A Formal Model of Program Dependences and Its Implication for Software Testing, Debugging, and Maintenance," *IEEE Transactions on Software Engineering*, 16(9), September 1990, pp. 965-979.

[10] D. Richardson, T.O. O'Malley, C. Moore, and S. Aha, "Developing and Integrating ProDAG into the Arcadia Environment." *Fifth Symposium on Software Development Environments*, McLean, Virginia, December 1992, pp. 109-119.

[11] C. Satterthwaite, "The Maintenance of Operational Flight Programs." *Proceedings 11th IEEE Digital Avionics Systems Conference*, Seattle, WA, October 1992, pp. 388-393.

[12] S. Walters, and M. Stephenson, "Virtual Test Station (VTS)." *Proceedings IEEE NAECON*, Dayton, OH, May 1993, pp. 965-972.

A FEEDBACK MODEL FOR THE SOFTWARE LIFECYCLE FOCUSED AT THE OPERATIONAL AND MAINTENANCE PHASE OF THE CYCLE

Charles P. Satterthwaite

WL/AAAF-3 Bldg 635
2185 Avionics Circle
Wright-Patterson AFB OH 45433-7301
E-Mail: sattercp@aa.wpafb.af.mil

INTRODUCTION

The software lifecycle continues to be the focus of much interest as more demand is placed upon computers and embedded computers to replace and enhance all features of life. Several models exist to describe the software lifecycle, all bringing a certain amount of understanding to the application, but none being robust enough to answer every application. This paper presents a project which duplicates the more common software lifecycle models with an operations research system dynamics approach. These models include the Waterfall, the Rapid Prototype, the Spiral, and the Fountain Models of Software Lifecycle. In all cases, the author has expanded each model to include a frequently neglected phase of the software lifecycle called operations and maintenance. It is the intent of this project and this paper to gain understanding as to how various parameters of the software lifecycle could be adjusted to improve the efficiency of the cycle. Current large software projects predict that their operation and maintenance costs are approximately 70% of the lifecycle costs. The purpose of the project is to look at how to effect this 70% as far as causes and what could be done to reduce it. The models basic unit is requirements, which could as easily be functions, man-hours, or even lines of code. [HAT88][KNI94][SOM89].

ABSTRACT

The statement of the problem is that the cost of software has been increasing exponentially as it has been used to add functionality and flexibility to almost every technology under development or in everyday use. As software process models are developed to explain and facilitate the use of software, the important phase of operation and maintenance of the software is often left out of these models leaving the process models incomplete. Additionally, the tendency to use software as a quick fix for system level problems that arise in the system lifecycle, delays the system level understanding of the software as an established defined function of the system.

The importance of the problem can be seen in the proliferation of software lifecycle models which include the Waterfall, Spiral, Rapid Prototyping, and most recently the Fountain Model. These models are often useful in understanding subsets of the software lifecycle, but leave some phases unexplained. The increased use of software is another indication of the importance of the issues of Software Engineering, and in this case the importance of the operational and

maintenance phases of the software lifecycle. Another clue is the evolution of standards addressing software development and maintenance. Standards such as Mil-STD-2167A and the DOD's Ada Language clearly indicate a focused energy on the this problem.

In order to better understand how system level decisions impact the software lifecycle's operational and development phases, a model of the software lifecycle will be developed using Professional Dynamo to provide a feedback scenario for the lifecycle. Since the Software Lifecycle is complex, certain assumptions will be made narrowing the focus of this project, but providing opportunities for follow-on work. The model will be rigorous enough to provide insights into resources required to include operational and maintenance issues into various software lifecycle models. The model will also give clues into who should be involved in setting system level requirements. An accumulation of trade-off data focused at a small product type application and a large long life application shall be analyzed. The model and the model assumptions shall be exercised to determine suggestions for extending those assumptions and for expanding the usability of the model. [HEN91][HUM89][RIC81][SOM89].

THE SOFTWARE LIFECYCLE MODELS

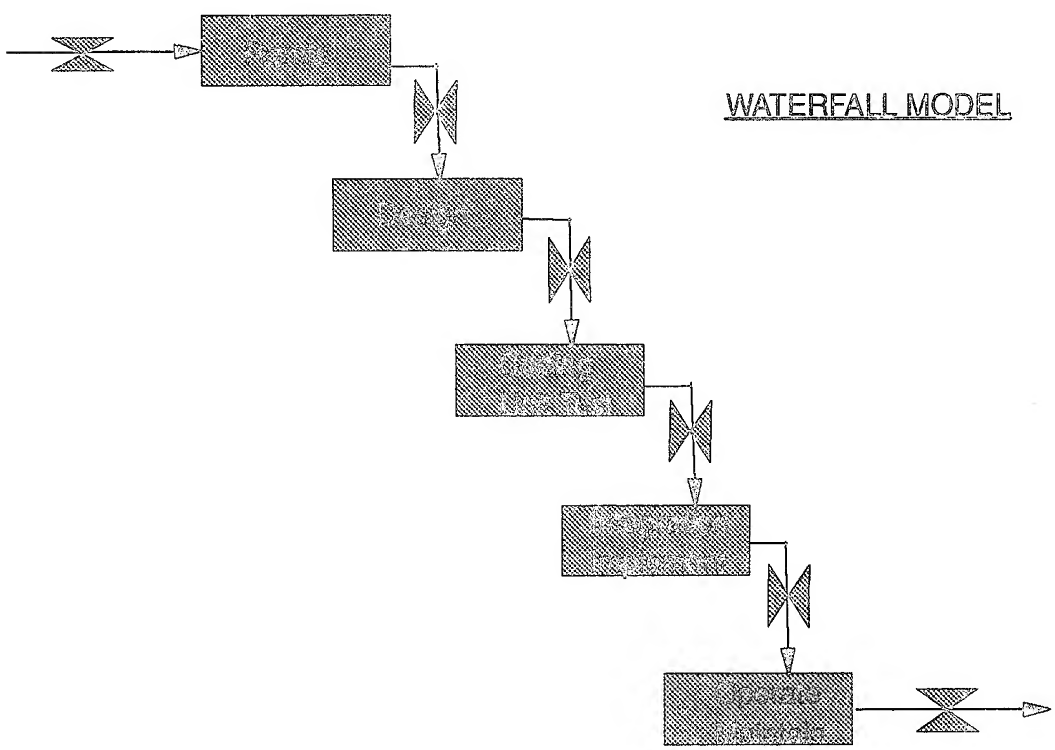


Figure 1 - The Waterfall Model

The Waterfall Model is the classic software lifecycle model which first attempted to explain the flow of software development through its various phases. As development finishes in a phase it flows to the next phase progressively until the final product is complete of the software

lifecycle within its system application. This and the other models represented in this project have been simplified to show an overview. Most software projects are broken out from their systems, and then reintroduced in the final integration of the system. The author believes this break out of software from its system is a fundamental flaw in System and Software Engineering. The author chooses to show an overview of a system with its software, with the following levels: requirements, design, coding and unit test, integration and implementation, and operations and maintenance as shown in Figures 1-4. The requirements and design levels are for both software

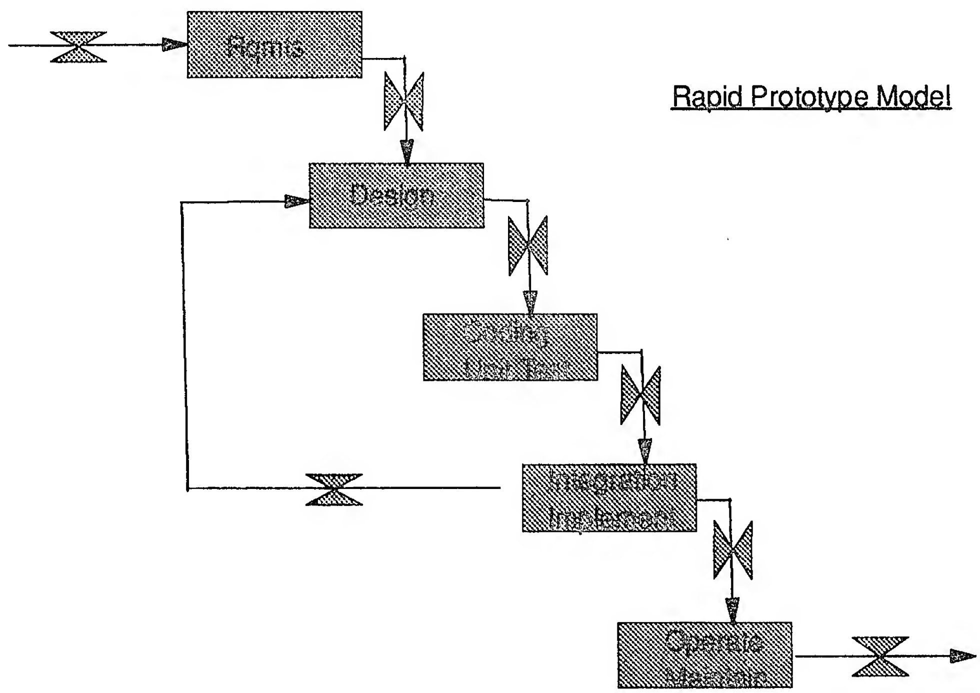


Figure 2 - The Rapid Prototype Model

and systems combined. The integration and implementation level assumes a continued integration of system and software issues throughout the software lifecycle. [SOM89].

The Rapid Prototype Model basically differs from the Waterfall Model in that a feedback mechanism is provided from the Integration an Implementation Level to the Design Level. The concept is to provide a quick turn around of prototypes for new design work. [SEG93].

The model in Figure 3 shows the Spiral Model, which provides feedback mechanisms from lower levels to the levels directly above them. The Spiral Model does not lock up any level of the software lifecycle. All levels are open to adjustments except the Operation and Maintenance Level, which can only influence the Integration and Implementation Level.

The Spiral Model is a frequently used software lifecycle development model, but like the proceedings models leaves little influence for the overall lifecycle on Operation and Maintenance interests. [SOM89]

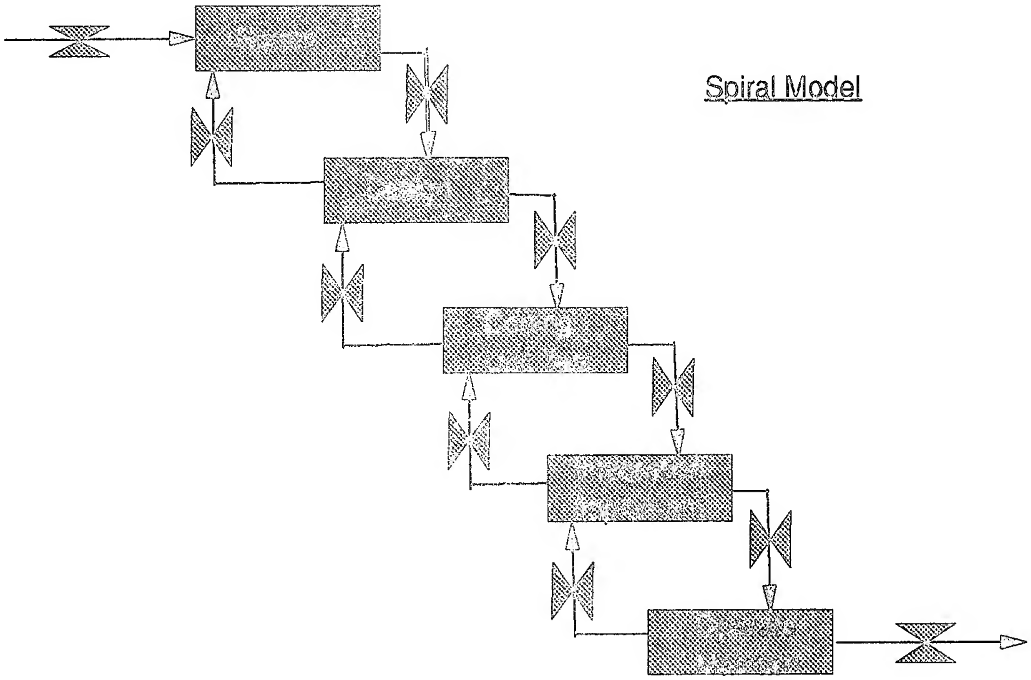


Figure 3 - The Spiral Model

The final Software Lifecycle Model that is being reviewed is the Fountain Model. The Fountain Model, unlike the preceding models, is the first model to seriously take into account the influences of the Operations and Maintenance Level of the Software Lifecycle. The Fountain Model provides feedback mechanisms from the lowest levels to all of its upper levels. Operation and Maintenance feeds back to Integration and Implementation, Coding and Unit Test, Design, and Requirements thus giving this level of the Software Lifecycle genuine opportunities to impact the overall efficiency of the lifecycle. [HEN91].

What is seriously missing is a means for which to adjust various parameters of the above models, and look at the results of these adjustments. Enough detail must be provided to accurately model the Lifecycle Modeling Techniques, but not to much detail to hide observations gleaned from the fine tuned adjustments. Another desirable feature, is that the models have enough common features so that the generalizations can be made across the models which can be further explored or detailed with follow on work.

This paper discusses a project which uses a systems dynamics simulation software tool called Professional Dynamo. This product is available commercially, and will be given a brief discussion. Professional Dynamo is a popular software tool used in the field of Operations Research. [RIC81].

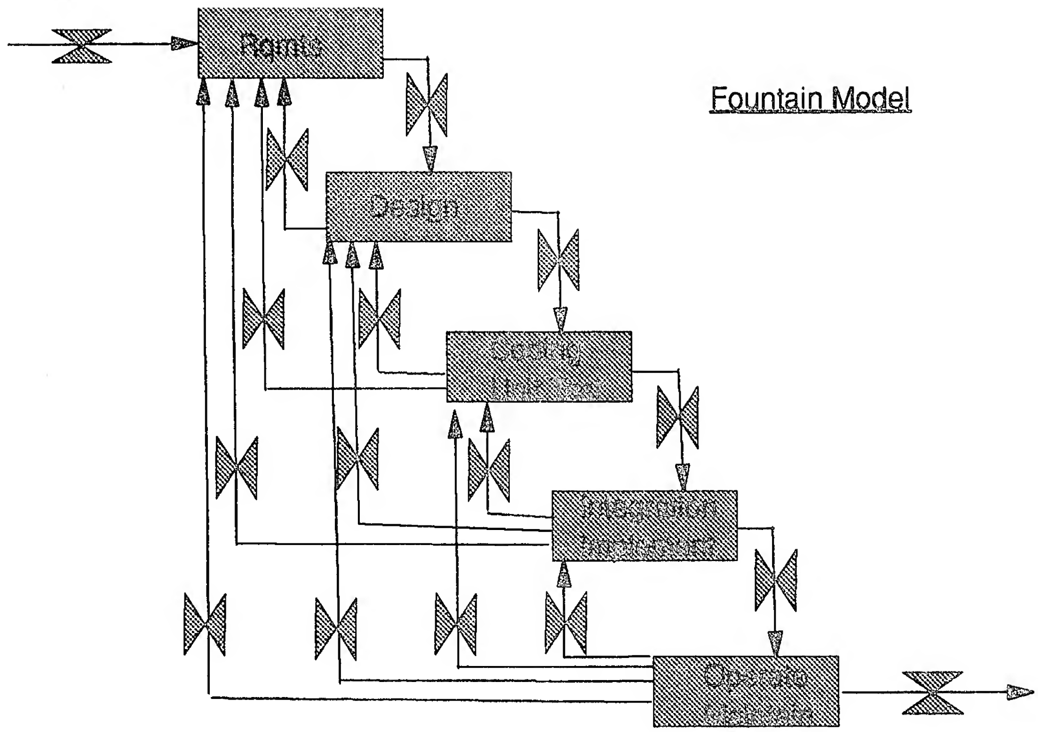


Figure 4 - The Fountain Model

PROFESSIONAL DYNAMO

Professional Dynamo is a systems dynamics simulation tool which models feedback systems. Most systems can be modeled by this type of tool, because they have some level of input from a source and output to a sink which flows through some type of state or states which are themselves effected by the systems environment. States can be modeled as levels of activity while flows can be modeled as rates, all of which drive or are driven by feedback of the environments parameters.

Professional Dynamo provides a software environment and simulation modeling language with rigor enough to model complex systems. Since the various software lifecycle models are complex, it is an interesting exercise to model them in Professional Dynamo. [RIC81].

THE SYSTEM DYNAMICS MODELS USING PROFESSIONAL DYNAMO

In order to represent the software lifecycle models discussed, Professional Dynamo is used to build a systems dynamic model for each case. Since the Waterfall Model forms a basis for each of the four models, it will be discussed in detail, and then each additional model's unique feature will be discussed. Additional sections will discuss assumptions, observations, and analysis of the four models.

The user of this model must first select which version of the Software Lifecycle Models to run. At this point these four models (Waterfall, Rapid Prototype, Spiral, and Fountain) are in a file format along with the Professional Dynamo Software. These files are accessed and edited to simulate the user's implementation from within the Professional Dynamo Environment. The user, upon entering any of these models, selects (by editing) a system type, a maintenance type, and a Tools And Methodology factor. System Types are: Realtime Avionics New (RAN), Realtime Avionics Heavily Modified (RAH), Realtime Avionics Low Modification (RAL), Realtime Vehicle New (RVN), and Nonrealtime Vehicle New (NVN). Maintenance Types are: Full (FUL), Normal (NOR), Minimal (MNM), and None (NON). Technology And Methodologies (TAM) values are: advanced environment = 2, well equipped = 1.5, moderately equipped = 1, and poorly equipped = 0.75. These types and values have been included to exercise the model, but are extensible to additional inputs.

Figure 5-9 illustrates the 5 levels of the Software Lifecycle Models and the inputs and outputs which determine those levels. As illustrated the inputs of many of the levels are the outputs from previous levels, so except for the case of the Operation and Maintenance Level, only the inputs will be discussed at each level. The basic unit of flow of this model is a requirement. A requirement is defined here as a needed functionality for the system which needs some or all of its implementation resolved in software. The more complex a system is in regards to its software requirements, the more requirements must flow through the models. [HAT88][HSI93]

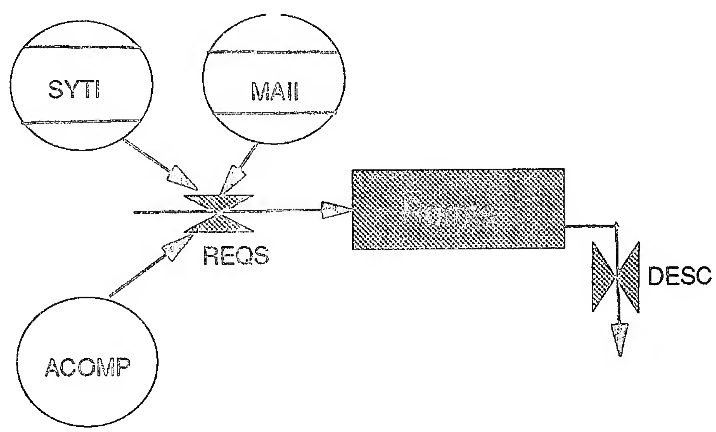


Figure 5 - The Requirements Level

The requirements level (REQ) is the difference between its input rate (REQS) and its output rate (DESC). The rate of requirements (REQS) is determined by adding table values of system type (SYTI) and maintenance type (MAII) which are inputted annually through the systems life, in this case 10 years. The sum of types is multiplied by an average complexity (ACOMP), added to 1, which is the sum of the following constants: modifiability (MODY), reusability (REUS), dependency (DPCY), sophistication (SOPH), documentation (DOCU), and configuration management (CONF) all which have a preset value of 0.04. [MUN92][PRI93].

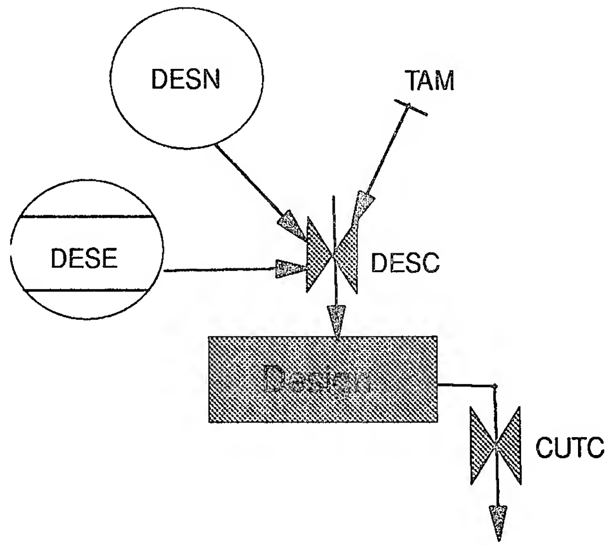


Figure 6 - The Design Level

The design level (DES) is the difference between input rate design capability (DESC) and output rate coding and unit testing capability (CUTC). The DECS rate is a product of design team experience table value (DESE), normal requirements handled in DES (DESN), and the tools and methodology factor (TAM).

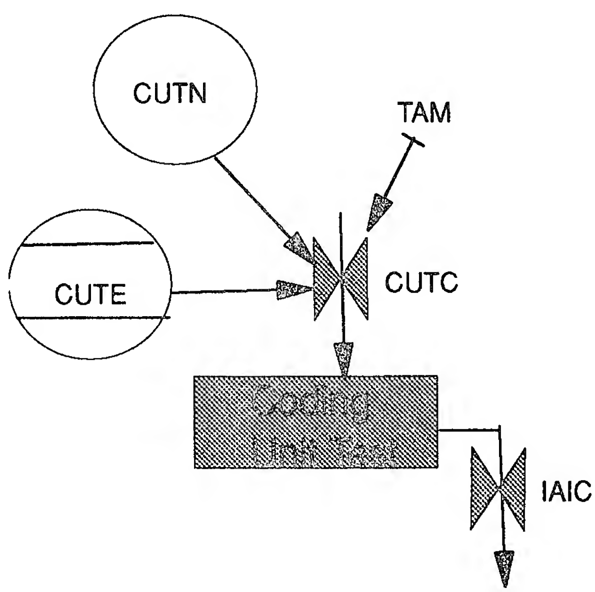


Figure 7 - Coding and Unit Testing Level

The coding and unit testing level (CUT) is the difference between it coding and unit test capability input rate (CUTC) and the integration and implementation output rate (IAIC). The CUTC is a product of the tools and methodologies (TAM), coding and unit testing experience table value (CUTE), and the normal coding and unit testing that design can handle (CUTN).

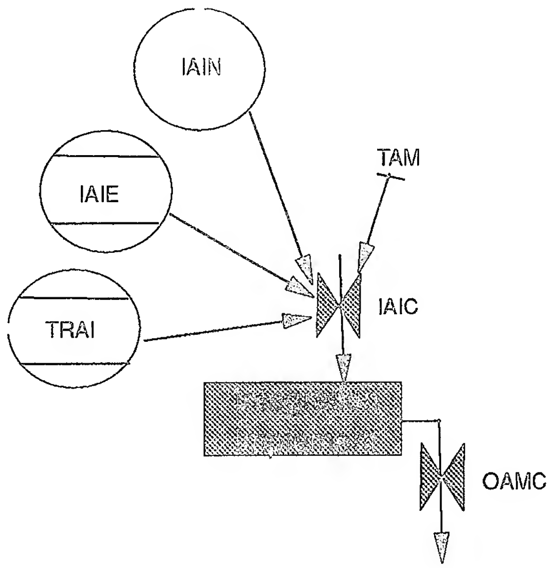


Figure 8 - Integration And Implementation Level

The integration and implementation level is the difference between its integration and implementation capability input rate (IAIC) and its operation and maintenance capability output rate (OAMC). The IAIC rate is the product of the tools and methodologies (TAM) factor, the normal integration and implementation requirements that can be handled (IAIN), the integration and implementation experience table value (IAIE), and the integration and implementation training table value (TRAI).

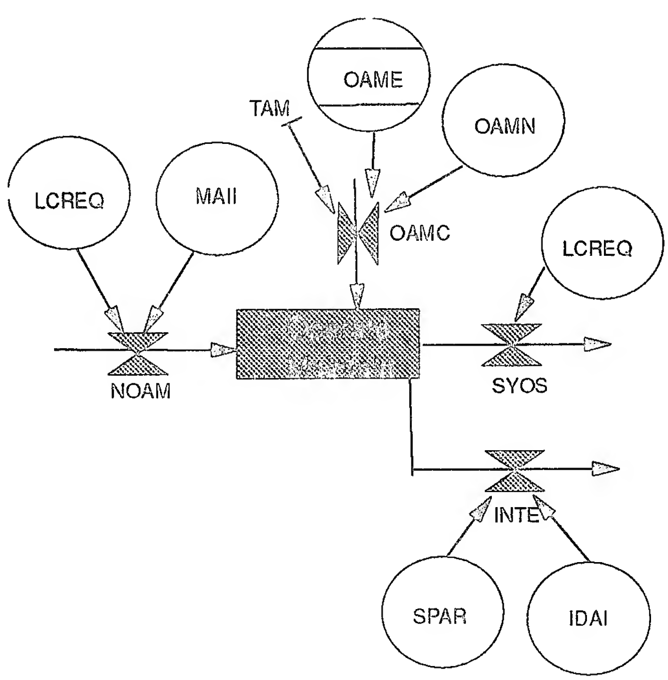


Figure 9 - The Operation and Maintenance Level

Figure 9 describes the level of activity most interesting to this paper. It is the operation and maintenance level, a level excluded from most software lifecycle models because of its complexity. The operation and maintenance level (OAM) has input rates from OAMC, previously described, and from a normal maintenance rate (OAMN). Its outputs are the systems out of service rate (SYOS) and the spares and technology rate (INTE). The OAMN input rate sums system type (SYTI), and the capability factors from design, coding and unit testing, integration and implementation, and operation and maintenance (LCREQ). Maintenance type (MAII) is subtracted from this sum and the result is multiplied by a factor of 0.3. The OAMC input rate, like the previous capability rates, is a product of tools and methodologies (TAM), normal IAI that can be handled in OAM, and the operation and maintenance experience table value (OAME). The systems out of service rate (SYOS) is determined by taking LCREQ and dividing it by the system out of service constant, which is in this case 10. The final output rate is the spares and technology rate (INTE) which is the sum of the spare part influence (SPAR) and the instrumentation and data analysis influence (IDAI). SPAR is the system type divided by 10 and IDAI is system type divided by 5.

The Rapid Prototyping Model differs from the Waterfall Model in that it introduces a feedback rate (IAIF) as an input from the Integration and Implementation Level to the design level.

The Spiral Model introduces feedback mechanisms from lower levels to proceeding levels. Each feedback is through a rate equation thus providing 4 additional rates which are design feedback rate (DESF), coding and unit testing feedback rate (CUTF), integration and implementation feedback rate (IAIF), and operation and maintenance feedback rate (OAMF). A change in this model over Waterfall and Rapid Prototype Models is that the Normal Maintenance (NOAM) is turned off. This is because a full feedback capability does not bear the penalty associated with the previous models.

The Fountain Model is the newest and most comprehensive of the four models. It provides feedback mechanisms from lower levels to all upper levels. Thus the requirement level (REQ) receives input from design, coding and unit testing, integration and implementation, and operation and maintenance feedback rates. The design level (DES) receives input from coding and unit testing, integration and implementation, and operation and maintenance feedback rates. The coding and unit testing level receives input from integration and implementation and operation and maintenance feedback rates. And finally, integration and implementation receives input from the operation and maintenance feedback rate. There are four additional rate equations as in the spiral model, only they have multiple applications as describe. Also, as in the Spiral Model, the Normal Maintenance Rate (NOAM) is turned off.

ASSUMPTIONS OF THE MODELS

Due to the complexity of modeling the software lifecycle and of generalizing 4 model types, many assumptions have been made. These assumptions are laid out in the same order as the model description followed, with a comprehensive discussion of the Waterfall followed by the unique assumptions of the Rapid Prototype, Spiral, and Fountain Models.

Assumption 1 is that the software lifecycle can be generalized to five levels of requirements, design, coding and unit testing, integration and implementation, and operation and maintenance.

Assumption 2 is that the average software life length is 10 years, and that requirements flow through levels is 1 year intervals.

Assumption 3 is that a large complex software dependent system can be represented by 200 requirements, released in intervals over 10 years, and that a small software dependent system can be represented by 30 requirements released in intervals over 10 years.

Assumption 4 is that maintenance requirements loaded with system requirements and released over 10 year intervals can reduce the operation and maintenance requirements burden.

Assumption 5 is that a tools and methodologies (TAM) factor can be set which captures an organizations ability to manage the software lifecycle. This assumption uses the same TAM factor at each level effected.

Assumption 6 is that an average complexity can be calculated by giving weights to the technical areas such as modifiability, reusability, dependency, sophistication, documentation, and configuration management. This assumption gives a weight of 0.04 to each and sets forward that the sum value of 0.24 captures the average complexity of all the systems.

Assumption 7 is that an input rate to each level of capability can be calculated by multiplying TAM, a normal handling value, and an experience value, and if applicable, a training value.

Assumption 8 is that each level can be assigned a normal value of what it can handle from preceding levels.

Assumption 9 is that each level gains experience annually throughout the life of system. An improvement factor of 1 percent is added each year for each level.

Assumption 10 is that in the integration and implementation level, an annual gain for training of 1 percent begins to occur in the 3rd year of the system.

Assumption 11 applies to only the Waterfall and Rapid Prototype Models. An input rate called normal maintenance is the product of 0.3 times the difference of the preset maintenance types from the lifecycle requirement factor. This is based on bringing a large systems operation and maintenance level to around 70% of the lifecycle requirements.

Assumption 12 is that 10% of the lifecycle requirements go out of service annually.

Assumption 13 is that there are 10% available spares annually, to take the place of incomplete requirements.

Assumption 14 is that 20% of the incomplete requirements can be handled with instrumentation and data analysis techniques annually.

Assumption 15 is that for the Rapid Prototyping, Spiral, and Fountain Models, the feedback rates are 20% of the levels they are being sent from.

OBSERVATIONS

In order to present a representation of the data generated from these models based upon the assumptions listed, Table 1 and Table 2 are provided which summarize the following: Listed in column 1 are the 4 models each with a large system row and a small system row. Columns 2-6 make up the inputs of the 5 levels at the 10 year point of the system's lifecycle. Table 1 summarizes systems with no maintenance requirements included. Table 2 summarizes systems with high levels of maintenance requirements include. This is only a small set of the available data, but sufficient to give some insight. A more comprehensive analysis of the data and exercising of the assumptions is being performed, but is outside of the scope of this paper.

MODELS	REQ-LEVEL	DES-LEVEL	CUT-LEVEL	IAI-LEVEL	OAM-LEVEL
Waterfall - L	31.4	25.4	61.7	1.9	236.3
Waterfall - S	4.5	3.8	9.2	0.4	35.6
Rap Proto - L	31.4	28.5	61.8	5	233.2
Rap Proto - S	4.5	4.4	9.2	0.9	35
Spiral - L	111.8	97.3	47.1	-21.3	-17.1
Spiral - S	15.2	13.7	7.4	-2.7	-2.5
Fountain - L	-1	42.5	22.8	-20.4	-20.2
Fountain - L	0.5	6.6	4.2	-2.5	-3

Table 1 - Software Lifecycle Models with no maintenance requirements

MODELS	REQ-LEVEL	DES-LEVEL	CUT-LEVEL	IAI-LEVEL	OAM-LEVEL
Waterfall - L	39.6	32.2	78.1	2.5	288.48
Waterfall - S	12.7	10.6	25.6	0.9	87.7
Rap Proto - L	39.6	36	78.1	6.5	284.5
Rap Proto - S	12.7	11.9	25.6	2.4	86.2
Spiral - L	141.9	128.8	76	-1.1	-0.3
Spiral - S	45.3	45.2	36.3	17.5	14.3
Fountain - L	196.3	118.3	71.2	0.2	-4.3
Fountain - S	197.8	82.3	52.5	18	12.8

Table 2 - Software Lifecycle Models with full maintenance requirements

MODELS with No Maintenance	5 Levels Summed at Year 10	OAM as a % of the 5 Levels Summed	MODELS with Full Maintenance	5 Levels Summed at Year 10	OAM as a % of the 5 Levels Summed
Waterfall - L	356.7	0.66	Waterfall - L	440.9	0.65
Waterfall - S	53.5	0.67	Waterfall - S	137.5	0.64
Rap Proto - L	359.9	0.65	Rap Proto - L	444.7	0.64
Rap Proto - S	53.1	0.66	Rap Proto - S	138.8	0.62
Spiral - L	256.2	0	Spiral - L	346.7	0
Spiral - S	36.3	0	Spiral - S	158.6	0.09
Fountain - L	65.3	0	Fountain - L	386	0
Fountain - S	11.3	0	Fountain - S	363.4	0.04

Table 3 - 5 Levels Summed at Year 10, and OAM as a Percentage of that Sum

Tables 1 and 2 show significant increases in the Operation and Maintenance Level when full Maintenance Requirements are included for every model type and size. When taken as an overall percentage though, there is a slight decrease as discussed below.

Table 3 looks at how the various techniques tried, effect the reduction of the Operation and Maintenance Level Requirements. Negative values are considered in this case 0 values, so for both large and small cases of the Spiral and Fountain Models, the Operation and Maintenance Requirements Level was reduced to nothing. Only in the Small System Type of Full Maintenance Requirements added, did Operation and Maintenance show any values at all.

For Waterfall and Rapid Prototype Models, there seemed to be a slight improvement on the Operation and Maintenance Levels of 1% decrease for large Waterfall, 3% decrease for the small Waterfall, 1% decrease for the large Rapid Prototype, and 4% decrease for the small Rapid Prototype Model. Adding Maintenance had little or no effect ON the Spiral and Fountain Models other than showing up as 9% for the small Spiral with full maintenance and 4% for the small Fountain Model with full maintenance.

When looking at models with progressing levels of feedback mechanisms (Waterfall - 0, Rapid Prototype - 1, Spiral - 4, and Fountain - 10) it is interesting to note an upward push of requirements into the Requirements and Design Levels. Rapid Prototyping pushes Integration and Implementation Requirements directly into the Design Level as expected.

Another interesting comparison is between the Spiral and the Fountain Models across the levels. Both models take on more requirements in early levels as stated, but the Fountain Model appears to be absorbing more of those requirements as they progress through the system.

A BRIEF ANALYSIS OF THE INITIAL DATA

An interesting observation is that the Waterfall Model in the small system configuration and with full maintenance. Through the life of the system the maintenance requirements are almost double the system requirements. Even with this large front end loading of maintenance requirements, only a small gain is made in reducing the final Operation and Maintenance Level. This could be due to not enough incentive be given at the Operation and Maintenance Level for front end loading of maintenance requirements.

In reviewing the Rapid Prototyping Model against the Waterfall Model, little gain is made in going to Rapid Prototyping for large or small models with or without maintenance requirements. This technique does shift requirements into the Design Level, but has no advantage for reducing Operation And Maintenance Level Software Lifecycle Requirements.

For the Spiral Model, requirements are pushed up into Requirements and Design Levels. Negative values in Integration And Implementation and Operation And Maintenance Levels indicate additional capacity in those levels to handle additional requirements. Front loading Maintenance Requirements does not seem to be an advantage in this model.

The Fountain Model is heavily impacted by the addition of Maintenance Requirements at the front end. Operation And Maintenance Requirements have 4 feedback rate paths, thus giving the Operation And Maintenance Level good access to Requirements, Design, Coding and Unit Testing, and Integration And Implementation Levels. The Fountain Model, as the Spiral Model offers spare capacity to large systems with front end loaded maintenance requirements.

CONCLUSIONS

Assumption 4, that front loading maintenance requirements in large and small systems does not appear to offer an advantage to offsetting Operation And Maintenance Level Requirements. In order to model an advantage here, more work is needed in the Operation And Maintenance output rates to show an increased advantage for front loading maintenance requirements.

As currently modeled, the initial data indicates that both Spiral and Fountain Models, with their broad feedback mechanisms, are very useful in offsetting Operation And Maintenance Level Requirements.

Much work is needed to further test out the assumptions of these models. By testing each assumption against each model type, it is believed that these models could be a useful tool for understanding the complex applications of software lifecycles.

The models are extendable, in that data used to build the models, such as table values and constants can be tuned to the real data of an application.

REFERENCES

[HAT88] Hatley, D. J., Pirbhai, I. A., *Strategies For Real-Time Systems Specification,* Dorsett House Publishing, NY, 1988.

[HEN91] Henderson, Sellers, *A Book of Object Oriented Knowledge,* Prentice Hall, Englewood Cliffs, NJ, 1991.

[HSI93] Hsia, P., Davis, A., Kung, D., Status Report: Requirements Engineering, *IEEE Software,* pp 75-79, November 1993.

[HUM89] Humpreys W. S., *Managing The Software Process,* Addison-Wesley, 1989.

[JON91] Jones C., *Applied Software Measurement - Assuring Productivity and Quality,* McGraw-Hill Inc. New York, 1991.

[KNI94] Knight, J., Littlewood, B., Critical Task of Writing Dependable Software, *IEEE Software,* pp 16-20, January 1994.

[MUN92] Munson, J. C., Khoshgoftaar, T. M., Measuring Dynamic Program Complexity, *IEEE Software,* pp 48-55, November 1992.

[PRI93] Prieto-Diaz, R., Status Report: Software Reusability, *IEEE Software,* pp 61-66, May 1993.

[RIC81] Richardson, G. P., Pugh A. L., *Introduction to System Dynamics Modeling with Dynamo*, MIT Press, Cambridge Mass., 1981.

[SEG93] Segal, M. E., Frieder, O., On-The-Fly Program Modification: Systems For Dynamic Updating, *IEEE Software,* pp 53-65, March 1993.

[SOM89] Sommerville I., *Software Engineering,* Addison-Wesley, Workingham England, 1989.

QUALITY FUNCTIONAL DEPLOYMENT (QFD) - HELPS THE DETERMINATION AND PRIORITIZATION OF REQUIREMENTS FOR EMBEDDED AVIONICS APPLICATIONS

Charles P. Satterthwaite

WL/AAAF-3 Bldg 635
2185 Avionics Circle
Wright-Patterson AFB OH 45433-7301
E-Mail: sattercp@aa.wpafb.af.mil

INTRODUCTION

This paper is intended to introduce the concept of Quality Functional Deployment (QFD), a technique which has come out of the Quality Movement from Japan and is sweeping American Industry, and explore how QFD could assist in managing the proliferation of embedded avionics requirements. The paper briefly introduces the problem of runaway embedded avionics requirements and its associated complexity; introduces QFD; shows how to build the QFD House of Quality; expand QFD Matrices; discusses the use of QFD in Embedded Avionics Applications; and concludes with a brief discussion.

ABSTRACT

PROBLEM STATEMENT: As the complexity of software projects for embedded avionics applications increases, it becomes more and more obvious that new innovative techniques will have to be used to help manage those projects. Complexity issues include the size, level of integration, level of testing, service life, and emergency change process, to name a few. An additional issue that software projects face is the need to include all the organizations touched by software through its lifecycle. These include not just the acquisition organizations, but the organizations responsible for the continued operation and maintenance of the software.

QUALITY FUNCTIONAL DEPLOYMENT AS AN INNOVATIVE TECHNIQUE (QFD): A new technique sweeping American Industry is QFD. QFD is a Japanese innovation which came out of their Total Quality Initiative, and which came to the United States in the early 1980s. QFD is a series of matrices built by crossing a list of whats (customer requirements) against hows (technical resources needed to accomplish a what). All of the parties involved in providing a customer service are involved in prioritizing the weights associated with the whats against each how. When the process of negotiating these weight is finished, a justified set of achievable requirements can be handed to a customer.

QFD IN EMBEDDED SOFTWARE PROJECTS: Like manufacturing processes, software projects touch many organizations. All to often one organization makes decisions which exclude the inputs from other effected organizations. When a software practitioner is fortunate

enough to bring the right community of effected organizations together, all to often he does not have a tool to capture and prioritize that community's desires. QFD provides that important tool, and as will be discussed in this paper, has been successfully used in embedded avionics software applications.

THE GROWING COMPLEXITY OF EMBEDDED AVIONICS REQUIREMENTS

The relatively new field of avionics is greatly improving aircraft performance by continuously introducing improved sensory suites, embedded processing , communication capabilities, integrated diagnostics, integrated support environments, and advanced architectures to name a few. Performance gains expand aircraft mission capabilities, aircrew survivability and safety, aircraft turn time, the adaptability of aircraft and crews to evolving threats, maintenance capabilities, and the overall understanding of aerodynamics.

There is a cost associated with performance gains. This cost is not easily quantifiable in terms of dollars, but is manifested in such areas as overall system complexity, organizational diversity, and the system lifecycle. System complexity is the easiest to understand. System complexity is seen in many forms such as exponential rise in embedded software, increased integration of sub-systems, the science of handling switches and displays, system test at various levels, and the number of technical personnel needed for the system to become a reality. Organizational diversity takes into account who is involved with the system. The system is procured, maintained, and operated. Any one of these might have multiples of levels of functional groups who have some influence on the system. A very difficult cost to understand is that of performance gain costs on system lifecycle. Though it is desirable to add capability, far to often their are costs associated with that capability far in excess of the installation costs. Training, spare parts, obsolescence, and sub-system interference are a few types of costs that could be uncovered when doing a lifecycle cost analysis.

The overall tasks of individuals responsible for setting the system requirements for embedded avionics applications is becoming increasingly complex because of performance gains available to those systems and because of the costs associated with implementing the performance gains. These requirements setting individuals need revolutionary new ways to deal with their responsibilities. [BRA94].

QUALITY FUNCTION DEPLOYMENT

When observing the Quality Movement in the United States, Japan, and Worldwide, it becomes apparent that there exists many opportunities to apply research and improve processes. Quality Functional Deployment, Concurrent Engineering, Just In Time Prototyping, and Rapid Prototyping are just a few of the many areas the Quality Movement is impacting. [BIS93] [RIC92] [ROS91].

Within the last 15 years the worldwide corporate community has been undergoing a transformation in recognizing the key elements contained within organizations who have a substantial reputation for producing quality products and services. The elements include a close

communication with customers and accurately assessing their present and potential needs, converting those needs into product specifications with consideration for present process capability within a industry, and continually improving the processes critical to an industry while providing feedback to the design process on breakthroughs that can improve the effort to exceed customer requirements. These organizations both surprise and delight their customers with the products and services they offer and insure the viability of their organizations by considering total economic impact of the actions taken within the organization.

The recognition of these elements worldwide can largely be accredited to the Japanese systemization of these elements during their national effort over the last 45 years in implementing the scientific method to manufacturing as outlined by Walter A. Shewhart and advocated by Dr. W. Edwards Deming. The previously mentioned key elements were repeated time and again within Japanese organizations as they competitively overcame Western industries in a wide array of manufacturing areas including most significantly electronics and automotive industries. [DEM86]

One of the analysis and implementation techniques that developed from Japan's national effort was Quality Functional Deployment. It was first conceived by Yoji Akao during his efforts to have production units understand the planning process necessary for quality assurance before producing new products. It was first implemented in 1969 at the Kobe Shipyard of Mitsubishi Heavy Industries during the design and production of supertankers. The technique was developed into a comprehensive system for manufacturing while Mr. Akao was chairman of the Japanese Society of Quality Control's QFD research committee from 1975 to 1986. QFD introduction into the United States can be traced to a series of seminars conducted by Yoji Akao in 1986 and was further enhance by Bob King's "cookbook" conversion of the technique, detailed in the publication "Better Designs in Half the Time". [KIN89]

Theoretically, QFD is a cross functional tool which can enable organizations to prioritize customer demands, develop reliable, cost effective and innovative responses to those demands, and orchestrate implementations of those responses. Functionally it is a series of matrices and charts that track and prioritize tradeoffs involved between different sets of related variables within the product development and manufacturing planning processes. This matrix is one of thirty proposed by Yoji Akao and prioritizes quality characteristics in relation to how they are impacted by customer demands. Matrices and charts can be picked from the field of thirty to serve the following purposes: listening to voice of customer, improving horizontal communication, prioritizing improvements, targeting cost reductions, targeting reliability, targeting engineering breakthroughs, orchestrating engineering breakthroughs, improving communication between design and manufacturing, and accessing process reliability. [KIN89] [IND93]

Successful implementation of these matrices within an organization is directly related to the accuracy of input, a focused effort on matrix calculations, and ability to interpret and communicate the meaning of the results. Coordination of these activities has typically been performed by one or a few individuals within an organization. King's cookbook conversion of the technique highlights the algorithmic nature of the matrices in calculation and shifting of information from one area of a chart to another and between related charts. It was theorized that

a successful computer software would relieve some of the repetitive mechanics within the process and enable the coordinators more time to focus on obtaining valid input from the organization and communicating the results of the matrices to the concerned organization. [KIN89]

BUILDING THE HOUSE OF QUALITY - THE FIRST MATRIX

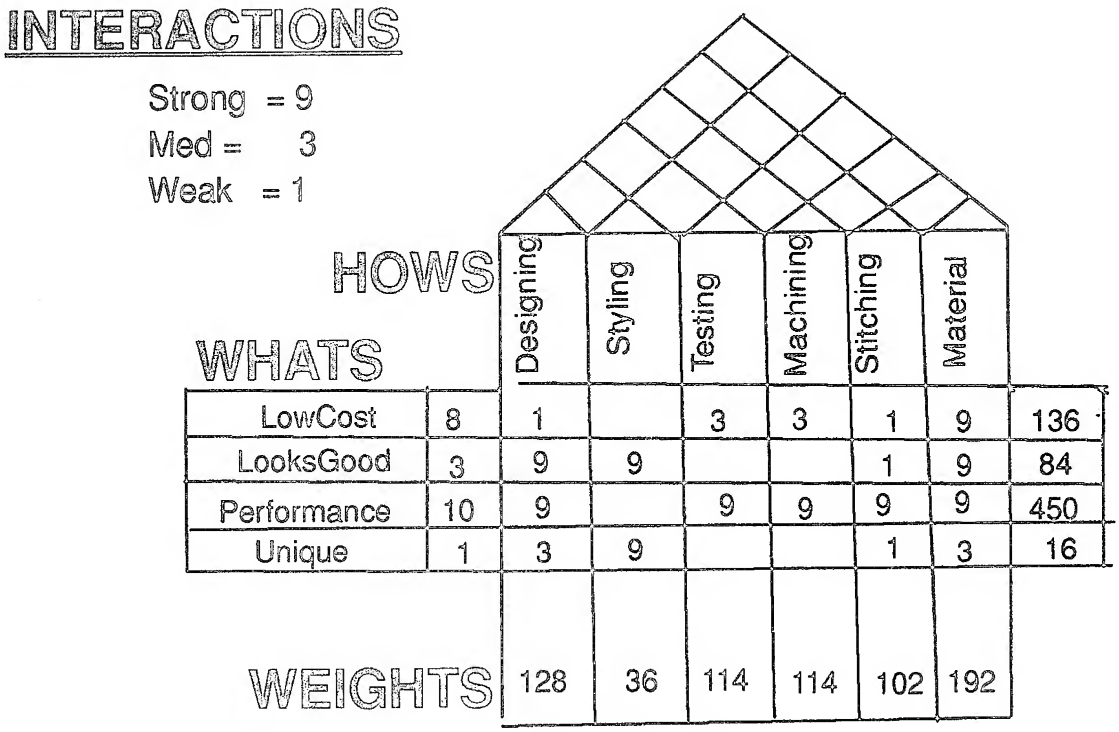


Figure 1 - The House of Quality

The House of Quality (illustrated in Figure 1) in and of itself greatly expands the ability to understand the requirements of a project (or a customer); associate those requirements to the tasks it will take to accomplish them; and weight the priority of the requirements in regards to accomplishing them.

In the simple example in Figure 1, the QFD team determines that in building their product, it has 4 requirements which are the WHATS. They are that it must be Low Cost, it must Look Good, it must Perform Well, and it must be Unique. To accomplish these requirements the tasks of Design, Styling, Testing. Machining, and Stitching must be accomplished. These are the Hows. The Whats are given Relative weights from 1-10 as they relate to each other. Weights of Whats against Hows are scored 9 for strong correlation, 3 for medium correlation, 1 for weak correlation, and empty (0) for no correlation of Tasks against the Requirements. For example, the Material is strongly correlated to Low Cost, and Performance. The sums of the Relative Weights times the Correlation Weights are tallied for each

. row and for each column to give a ranking for the Whats along the rows and the Hows along the columns. For our example row ranks were Performance, Low Costs, Looks Good, and Unique. Column ranks were Material, Design, Testing, Machining, Stitching, and Styling.

Though this is a simple scenario, this same approach can be used to capture and evaluate a multitude of requirements against a multitude of technical tasks. Given the tools now available to automate the matrix calculations, the QFD Team can use their efforts to exhaustively list out their requirements and technical tasks, agree upon the relative weights and the correlation weights, and interactively agree upon tradeoffs on their requirements as the QFD System gives them results.

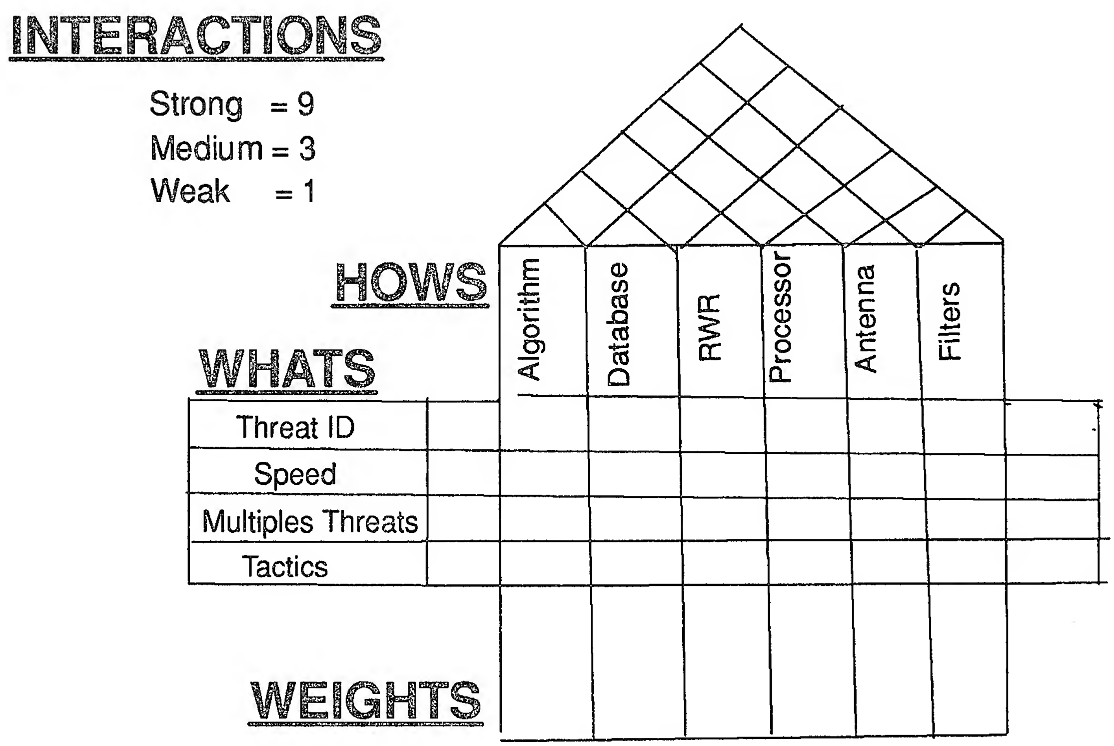


Figure 2 - The House Applied to a Simple Electronic Warfare Application

Figure 2 allows us to fill in the squares and do a simple QFD House of Quality interactively. This exercise has assumed 4 requirements of Threat ID, Speed, Multiple Threats, and Tactics. The technical tasks or hows are Algorithm, Database, RWR, Processor, Antenna, and Filters. This exercise is to first set relative weights of requirements, set the correlation weights and sum the results.

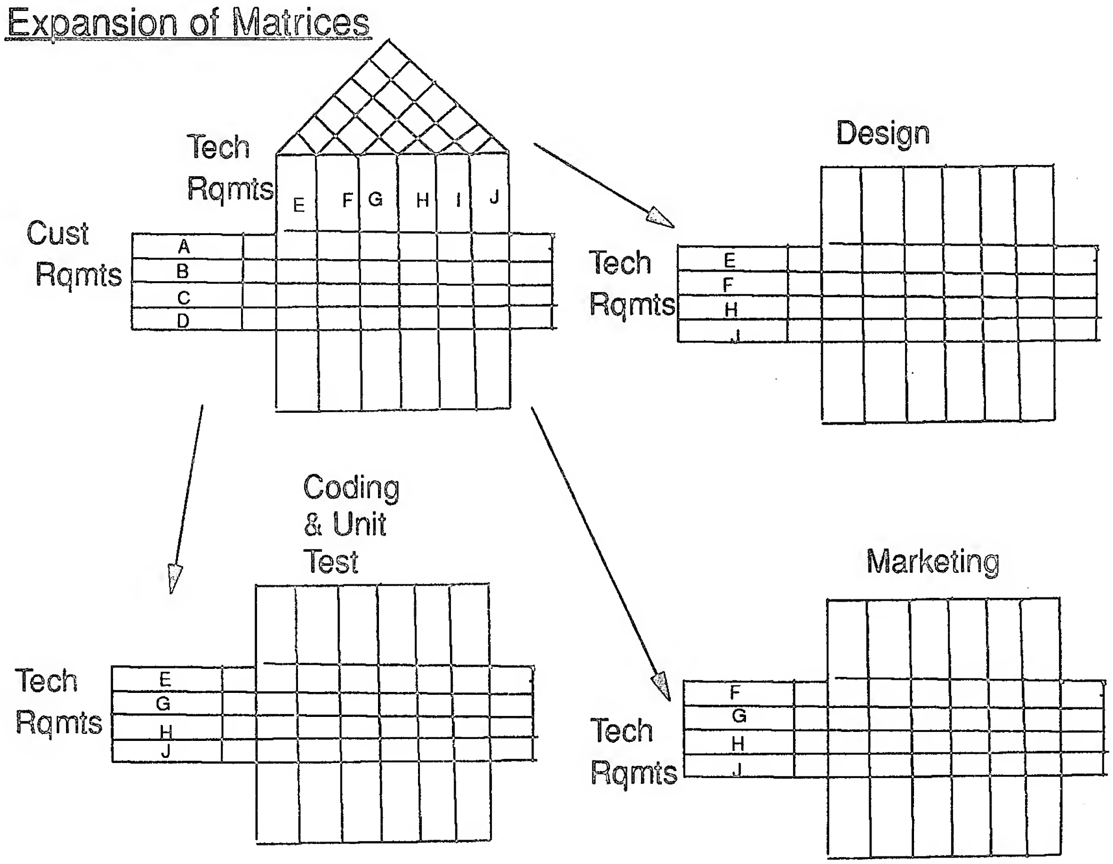


Figure 3 - The Expansion of Matrices

QFD is not limited to the House of Quality examples, though these are powerful tools. The House of Quality is the initial matrix, which captures the customers requirements. The technical requirements coming out of the house of quality become WHATS to another matrix which might take the Technical Requirements to Design, or to Coding and Unit Test, or to Marketing as illustrated, or any other arrangement the organization would want implemented.

In its full implementation, QFD could have 30 Matrices, all interconnected to give a technique for tracking and measuring every aspect of your product or service as it goes from concept (cradle) to grave.

QFD IN EMBEDDED AVIONICS PROJECTS

QFD is being successfully used by Hughes Aircraft Company to support several United States Air Force projects at Robins Air Force Base involving F-15 and AC-130U radar support. Hughes worked with the Air Force to ensure that a mix of individuals attended the system requirement review (SRR) of these projects. These individuals represented the various interests associated with developing and operating the various radar support environments. These included radar operational flight program (OFP) developers and maintainers, radar support

environment engineers, testing organization representatives, and program managers from the system program office as well as the integration support facilities. Finally, there was the Hughes contractors.

The author was invited to facilitate one of these QFD activities and was the program manager of another. These insights are from the authors experiences to offer a brief glimpse at a successful implementation of the QFD Process into setting Embedded Avionics Requirements.

Hughes first task in doing QFD, was to assemble a QFD Team This team's role was to set the requirements of the project. Traditionally, this team was confined to the contractor and the customer of that contractor. Outsiders who had big stakes in the outcome of the project seldom were asked to participate. Hughes changed this by opening the doors to all stake holders. With the team assembled, it takes an experienced facilitator to make sure the overall objectives of the project are reached and that no party dominates the setting of the requirements, tasks, and weights.

Once requirements are exhaustively listed, they must be given a relative ranking against each other. Her is where the emotional and political action takes place. One person's relative weight of 10 is anther's weight of 1. A compromise of a 5 might resolve the conflict, or the 10 might be held out for. But in the end a quorum is reached which reflects a healthy cross section of organizations, rather than the earlier narrow cross section.

Determination of the correlation weights is smoother than relative weights. Sometimes the combined expertise of the QFD Team makes this part of the process enjoyable, by introducing the team members to new perspectives of what it takes to accomplish their interests. For example, here is where the software programmer becomes aware of the cabling requirements for his new super X computer.

In one of Hughes projects with Warner Robins, 200 requirements were sorted out and prioritized. The results of the QFD exercise helped establish a successful support environment which will soon be operating in support of Warner Robins F-15 Avionics Integration Support Facility.

CONCLUSIONS

It is increasingly complex to manage embedded avionics systems. The new technologies being made available and the appetites of system operators for performance and capability gains place continued pressure on systems acquisition, development, and maintenance concerns to find new ways to perform their functions and to perform them more efficiently and jointly.

QFD offers a powerful approach for understanding requirements and for working together. Beyond the House of Quality, QFD can also trace and measure the attributes of making requirements capabilities.

QFD Software products are available which allow the process to be automated. It is strongly encouraged to take advantage of automation. This allows the time spent in QFD activities to be focused on Requirements setting and trade off analysis, rather than manipulating the multitudes of calculations in the matrices. Also this provides almost instantaneous feedback to the participants as they determine and prioritize the requirements of their embedded avionics applications.

REFERENCES

[BRE94] Brecka J., The Biggest Ever: Quality Progress' 11th Annual QA/QC Software Directory, *Quality Progress*, pp 60-115, March 1994.

[BIS93] Bisgaard S., Statistical Tools for Manufacturing, *Manufacturing Review*, Volume 6 Number 3, pp 192-200, September 1993.

[BRA94] Bradley D., Satterthwaite C., Quality Functional Deployment (QFD) - An Assessment of Implementation and Software, University of Dayton graduate project for Engineering Management 582, July 1994

[GIB94] Gibson J. L., Ivancevich J. M., Donnelly J. H., *Organizations*, Irwin, Burr Ridge, Ill., 1994.

[HEN91] Henderson, Sellers, *A Book of Object Oriented Knowledge*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[HUM89] Humpreys W. S., *Managing The Software Process*, Addison-wesley, 1989.

[JON91] Jones C., *Applied Software Measurement - Assuring Productivity and Quality*, McGraw-Hill Inc. New York, 1991.

[KIN89] King B., *Better Designs in Half the Time - Implementing QFD Quality Functional Deployment in America*, GOAL/QPC, Methuen, MA, 1989.

[RIC92] Richa C., Edwards D., A JIT Implementation Plan Using Policy Deployment, *Manufacturing Review*, Volume 5 Number 3, pp 166-174, September 1992.

[ROS91] Rosenblatt A., Watson G., Concurrent Engineering, *IEEE Spectrum*, pp 22-37, July 1991.

[SOD89] Sodhi J., *Managing Ada Projects Using Software Engineering*, TAB Books Inc. Blue Ridge Summit PA, 1989.

[SOM89] Sommerville I., *Software Engineering*, Addison-Wesley, Workingham England, 1989.

An Automated Approach to Maintaining Embedded Software Legacy Systems

David F. Tyler and Steve Whiseant
Access Research Corporation
Division of TYX Corporation
1851 Alexander Bell Dr.
Reston, VA 22091-4384 USA
Phone: (703) 264-1080 Fax: (703) 264-1090

Abstract: This paper addresses a current and future requirement to maintain embedded software legacy systems. Throughout the entire DoD there are critical real-time, embedded software systems that have been developed prior to the current focus on configuration management (CM) and Computer-Aided Software Engineering (CASE) environments and practices. These systems can be reverse engineered and brought into a CASE or maintenance environment in a semi-automated manner with all of their capabilities properly and fully defined and preserved. Once a baseline has been established, they can be properly maintained by the using and maintaining organizations according to the best commercial practices. In a CASE environment, the maintainer can re-engineer the system to accommodate major changes.

1.0 INTRODUCTION

Re-engineering of fielded systems will eventually be required as hardware platforms and software operating systems inevitably become obsolete. Re-engineering is also required if the system must be significantly upgraded for application specific reasons. This paper provides an overview of an established technique called Computer Aided Re-Engineering (CARE) for performing these tasks in a semi-automated manner. The benefits of re-engineering any system can be categorized from two perspectives:

Managerial	Engineering
• Configuration Control (CM)	• Overall Visibility
• Reduction in Cost and Schedule	• Factual Findings
• Risk Reduction	• Detailed Design-type Database

From the managerial aspect, a most critical element of Program Management (particularly with regard to Life Cycle Cost (LCC) and the maintenance portion of system support) is CM. One of the most useful elements of CARE is to baseline an existing product. There are three main baselines:

1. **Product Baseline** - As built description in terms of function, performance and operational characteristics.
2. **Allocated Baseline** - Re-manufacturing of detailed functional design specifications and performance verification of the existing system.
3. **Software Functional Baseline** - Software specifications are made compliant with existing capabilities.

The CARE process begins with the establishment of the Product Baseline. As a minimum the user can maintain the system from this baseline. Once the Product Baseline is obtained, the engineers develop an Allocated Baseline and finally the Software Functional Baseline. It is from the Software Functional Baseline that forward engineering can commence. The benefit of this is that Program Management can know exactly what they do and don't have from the existing product, i.e., risk can be accurately ascertained from the software functional baseline. Program Management can present the using organization(s) with what risks lie ahead with a higher level of confidence, especially when compared to not having any baseline.

From the Engineering perspective, the hardware and software engineers can obtain an overall system knowledge in a time-efficient manner. When computer aided methods are used, engineers have access to accurate factual findings in a detailed database. At this point, the system requirements are virtual and can be dealt with in a variety of methods and are not language dependent. Currently, the CARE method proposed here has ten major process elements for the completion of the entire re-engineering process. Once the baselines are achieved, the user of the technique has the complete processing capability to begin forward engineering in CASE environment.

2.0 The Need For Re-Engineering

The need to re-engineer real-time embedded systems that have been in use is increasing on a day by day basis. Today, most airframes cannot perform their missions if there are major Operational Flight Program (OFP) troubles, and in some cases the airframe is not even airworthy given certain avionics or control system anomalies. While there have been significant gains made in the software development technology that will be used in future OFP development programs, the majority of systems currently fielded have proven to be nearly un-manageable from a maintenance perspective. The DoD agencies responsible for maintaining the equipments they manage have to rely on the prime contractor for OFP support for the entire life cycle. This is expensive, not usually very timely and even a major problem for the prime contractor at times.

This problem arises because there are essentially very few engineers that know the OFP and its related hardware well enough to make significant changes. Another problem that arises after a program begins to age is that the best engineers migrate to new and more exciting programs. As the DoD takes further control of the management of the OFP, this problem becomes exacerbated. The need for re-engineering becomes obvious: How can application specific knowledge be transferred to the maintainer? Re-engineering is one answer.

There has been a movement afoot for the past several years for the responsible DoD agencies to obtain organic support for OFPs. This does not necessarily mean that there is a program office in the organization that is staffed purely with government employees, but rather there is a mix of support engineers from the prime contractor, service contractors and government personnel. This group is usually located at the maintaining organization's physical site. Changes can be made rapidly and management control is more easily maintained, but even here major technical problems exist.

Re-engineering provides a knowledge base for maintenance experts that can manipulate systems for maintenance purposes. This is very different in many ways to the group of engineers and designers that developed the system in the first place. The ability to have an organic software maintenance

capability has an even greater import when one considers the current trend of upgrading existing weapons systems and not replacing them with completely new designs.

3.0 AN OVERVIEW OF THE CARE PROCESS

CARE is a process that is made of both automated and manual processes. An overview of the entire CARE process is given in Figure 1. The intent of defining the entire process is to structure the program tasks into logical, functionally complete subtasks that can be reasonably managed. Program management can use this process chart as a road map that contains all of the individual subtasks that must be accomplished.

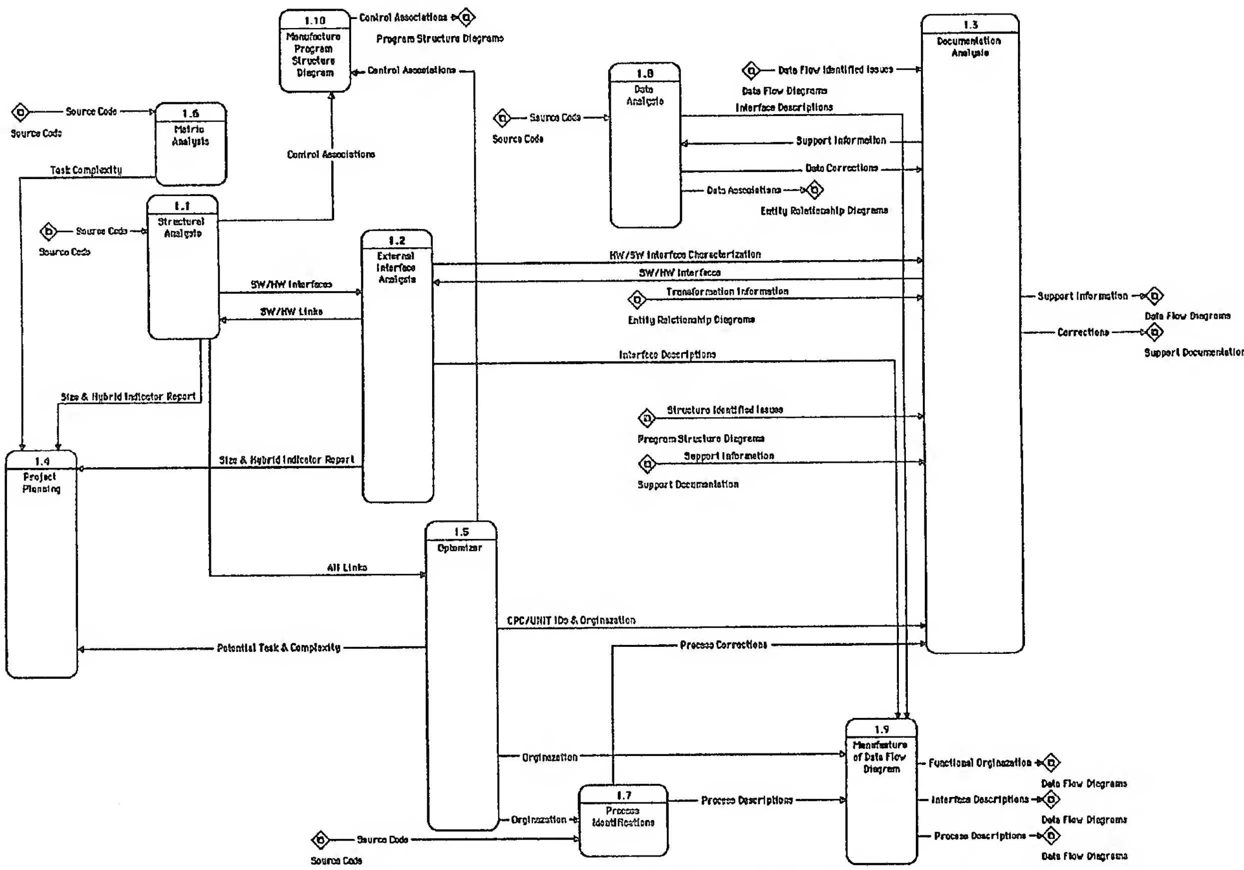


Figure 1. Overview of the CARE Process

3.1 Re-Engineering Metrics

In Process Block 1.6 of Figure 1 Metric Analysis occurs, wherein the source code is analyzed for its complexity. Metrics are a basis for the determination of the size of the total re-engineering job. Software metrics for real-time, embedded systems applications that CARE uses were developed from metrics that are typically used for standard software applications, where the real-time elements are either minimal or nonexistent. The development of software metrics for real-time systems required that many additional factors be considered, such as hybrid processes, state initiated machines, hardware interrupts and so on. The beginning of the metric development process was to develop

metrics that would assess the "understandability" of the system. This is accomplished by the identification and analysis of functional boundaries.

The identification of functional boundaries is based on the work of Miller, Parnas, Jones, Halstead and McCabe. Measurement of human comprehension is a primary focus of these metrics. Human comprehension is defined as the boundary of "understandability" of human engineers in a typical working environment, without causing excessive stress.

The area of contribution that the referenced works have provided are:

- Parnas - System decomposition theory
- Miller - Grouping and hierarchical boundaries of understandability.
- Jones - Function point understanding and requirements for successful support environment
- Halstead - Complexity boundary
- McCabe - Complexity boundary

Abstraction theory was critical in devising software metrics to accomplish computer-aided Re-Engineering. The initial concepts were based on the work of Parnas. Parnas has documented his approach to decomposition of existing systems and the building of abstracts. Functional compartments are groupings of code that serve some goal within the system. The hierarchical associations are developed from analysis of relationships and access requirements of each software entity and are based on Object Oriented software design guidelines and methods.

Once the functional boundary metrics were developed, the code specific complexity metrics were developed. This development took into account standard complexity metrics as well as extensions that accounted for real-time interfaces and other peculiar embedded systems aspects of the design. This is a much more straightforward approach to use when a system is decomposed into appropriate functional boundaries.

The metrics were applied to samples of OFP source code of varying degrees of complexity and those results were reported to another group of OFP engineers that analyzed them. The deficiencies in the metrics were noted and the metric developers then modified the models to account for those deficiencies. This process continued until there were little or no criticisms of their ability to identify functional boundaries and software complexities. After this, the metrics could be used to make complexity inferences on total software systems based on extensive internal samples.

3.2 Structural and External Interface Analysis

Process Block 1.1 of Figure 1, Structural Analysis, is where source code is read into the system and initially parsed with the CARE system. The Structural Analysis produces control associations for the manufacturing of Program Structure Diagrams (PSDs), a list of software and hardware external interfaces and all associations, both data and control, to be provided to the Optimizer function, Process Block 1.5. The Structural Analysis tools also make determination of the size of functions within the software system, and in cooperation with the External Interface Analysis tools, Process Block 1.2, identifies the existence of any hybrid functions in the system. These functions include event driven, interrupt driven and processor initiated state machine activities that impact the operation of the subject system's operation. The functional modes and states of the software and its related hardware system must be ascertained via existing documentation or by manual analysis.

These hybrid processes, including processor initiated state machines within source code, are highly dependent upon hardware and external interfaces. They cannot be parsed automatically and must currently be analyzed manually by computer system experts. The intent of the CARE technique is to focus the engineering staff's talents to concentrate on these areas of difficulty and to remove the mundane and tedious work from the engineer's concern. By focusing the talents of the engineering staff, significant gains can be made with respect to the efficiency of the re-engineering and maintenance process. These external hybrid functions are typically critical to the overall system operation and are the areas that are generally the least documented. Figure 2 illustrates the difference between the typical PSD and a CARE PSD.

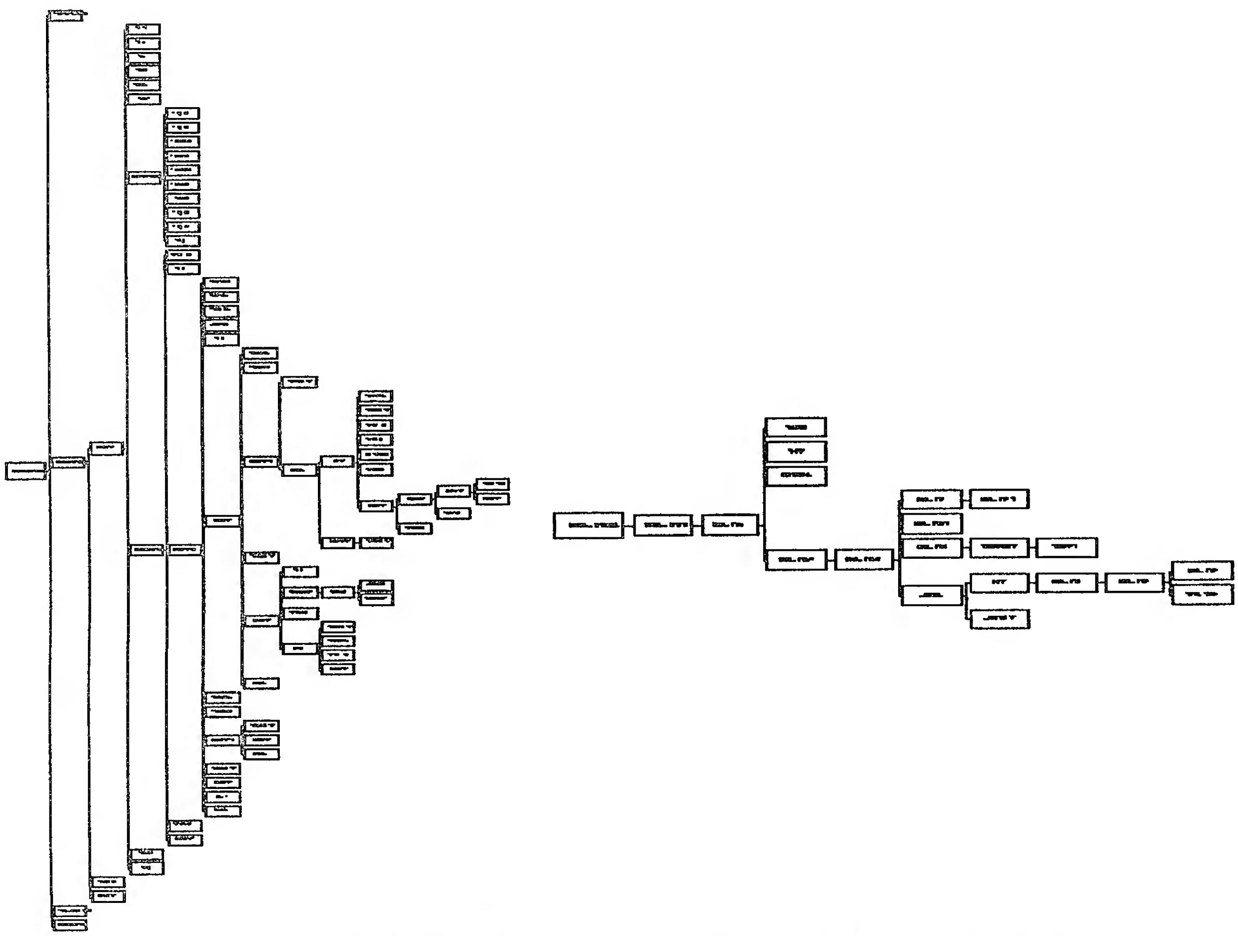


Figure 2. Typical PSD View of System vs. CARE View of Systems

3.3 Data Analysis

Process Block 1.8 of Figure 1, Data Analysis, is used to develop an understanding of how to accomplish data rationalization for the subject software system from the source code and any existing support documentation for the system. The Data Analysis activity enables the decomposition of the data flows to the lowest possible levels so that pertinent information is available at the location where it is most beneficial. Data rationalization is based on data abstraction theory and determines exactly how the subject system's data is to be represented in the maintenance or CASE environment. The Data Analysis process outputs Interface Descriptions for the manufacture of Data Flow Diagrams

(DFDs), Data Corrections for existing support documentation (or for inclusion into new documentation) and Data Associations for the development of Entity Relationship Diagrams (ERDs). To be able to create ERDs and DFDs from the automatic parsing of source code is of substantial utility to the maintainer, regardless of how much documentation is available.

3.4 Process Identification and Optimization

Process Block 1.7 of Figure 1, Process Identification, helps to develop a functional understanding of system operation and delineates boundaries around unique capabilities. The Process Identification algorithm requires input from the Optimizer process identified in Process Block 1.5. The Optimizer process provides an organization that is hierarchical in appearance and represents the system in the most concise manner possible. The process identification takes this perspective and groups the arrangement presented by the Optimizer into functional packages by application of Miller's principles of human understandability metrics. The development of these functional packages is a key link in establishing synchronization to the existing documentation or re-creation of the support documentation.

The user of the CARE process will have an excellent idea of the logical leafs and nodes of the system. In essence, they will know what is a functional entity and where that entity is placed in the system. The analyst will also be able to determine how the entity operates within the system and if that entity has been properly documented. Figure 3 contains an elementary functional boundary graphic illustrating this idea.

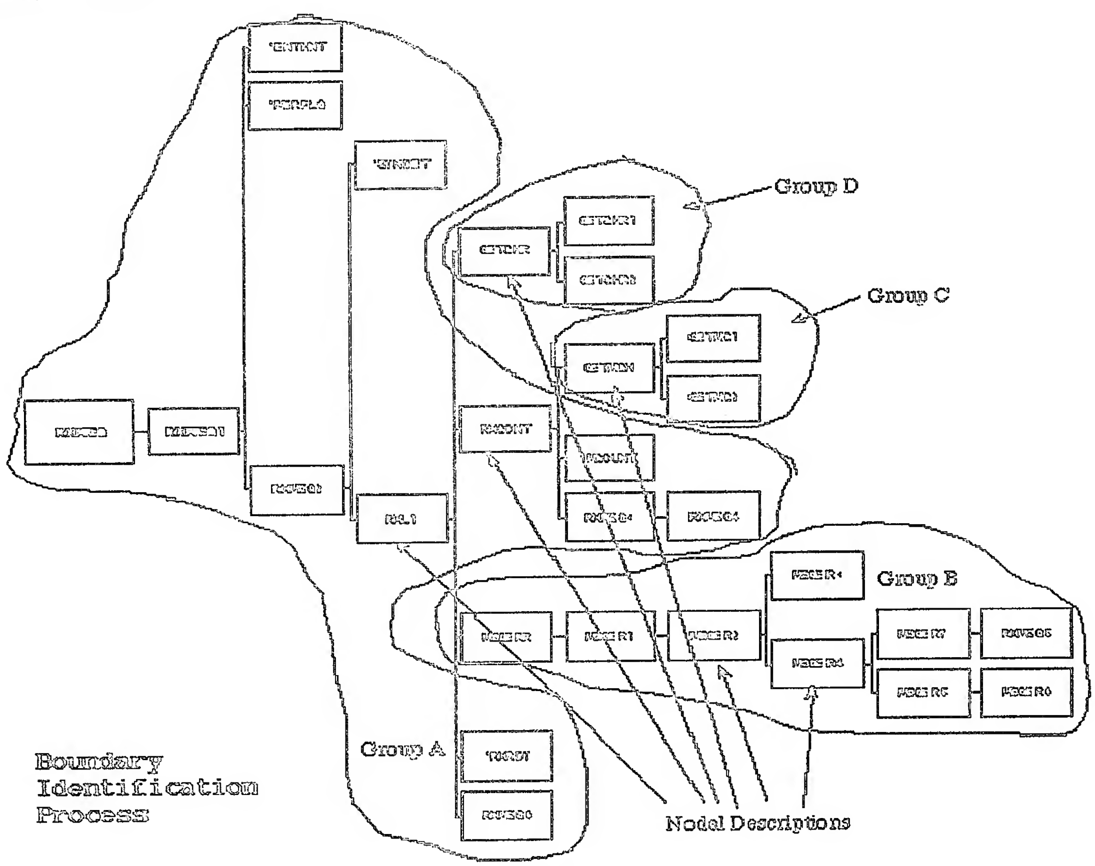


Figure 3. Functional Boundary Identification

These first three CARE processes use the source code of the system under analysis to begin the analytical process of re-engineering. They are interactive and part of an iterative process, so it is a dynamic solution that is carried out. Because it is dynamic, it is extremely flexible and powerful in solving the problems associated with re-engineering and related tasks of large software systems.

3.5 Documentation

The Documentation Analysis, identified in Process Block 1.5 of Figure 1, is a two aspect process that provides the user with an independent assessment of the software system's documentation as well as additional documentation that might not have been part of the original system development. In many cases that CARE has been applied to, the existing system was developed before the concept of using DFDs or ERDs to describe the architecture of the design. It is most common to find flow charts as the sole means of describing the designs of older software systems. This leaves much to be desired as data flow and other forms of structural information are required before the operation of a system can be truly understood. Flow charts give little or no information about the various operational states and modes a system can be in as well as almost no idea about the structure of data within the system.

The automated parsing technology of the CARE approach determines the structure of a system by "feeling out" all of the nodes, data paths and so on. It is not trying to present a "good" picture to a customer, but rather the "harsh reality" of the given design. From the maintaining organization's point-of-view, this is a better tack. There is much less risk in knowing than not knowing that the program tasks ahead are very difficult.

In past applications of the CARE process to real-time embedded systems it was common to find documentation that fell behind due to the constant upgrade and maintenance of the system. Time is frequently not allocated for the tasks of documenting what changes have taken place. With CARE, this issue can be resolved in a fairly automated manner. The more automated the process is, the more likely it is that it will be accomplished.

3.6 Project Planning

Process Block 1.4 of Figure 1, Project Planning is used to support the development of estimating tasks. This includes metrics such as time-to-accomplish tasks, partitioning of functional tasks, level of software engineer required to accomplish tasks, number of software engineers at each level required to accomplish each task and overall workload to establish a well thought out and reasonable time-line and schedule. From this time-line, accurate cost estimates can be generated and risk can be adequately ascertained.

The metrics measure uses pre-defined random variables such as how much time was saved, how many lines of code were developed, how complex the system is, how tightly coupled it is and so on. This gives the maintainers and users an estimate of how good its design is and what to expect for any significant modifications.

4.0 What About CASE For Real-Time Embedded Legacy Systems?

After application of the CARE process to several different software domains, from a maintenance of legacy code scenario, it has become obvious to its users that CASE, as it is currently marketed, is not

as viable as it is purported to be. Upon further investigation into this outcome, the reasons for this are understandable, viz.;

1. CASE has been for the most part designed for forward engineering applications. Reverse and Re-engineering applications demand various capabilities from CASE systems that violate the basic premises of those CASE systems and their designers.
2. CASE systems push a set of development procedures that are not intended to be broken. The design methodologies employed are typically structured, top-down approaches that limit the unstructured development of procedures, subroutines, modules and the like.
3. The very real need of being able to re-compile changes to the existing source code is not accommodated by any of the CASE environments currently in use.
4. CASE environments for assembly languages are not given much attention by the major manufacturers.

CASE has not been the answer for many real-time embedded legacy applications to date. In the real-time embedded systems world, there are many such applications that require millions of dollars of resources per year to maintain. In the current OFP scenario, the changes are tested in a "Hot Mock-up" and then brought to a test range and flown until a wide range of operational evaluations are completed. These operational evaluations are expensive and in some cases dangerous to execute.

While many newer systems are being attempted in the CASE environment, there has not been enough data to determine if even in a full forward engineering environment CASE will be effective. It is obvious that after years of maintaining real-time, embedded systems that being able to have tools that manipulate the source code and to have the ability to re-compile it within the maintenance environment is essential.

5.0 Summary and Conclusions

CARE is a viable approach for the for the development of the initial and subsequent baselines of existing systems as defined in this paper. While it is still a combination of manual and automated processes, it provides a structure to the often unstructured world of real-time, embedded systems maintenance. CARE is very effective in developing accurate documentation and a system's knowledge base. The CARE approach still requires systems engineers and programmers with considerable expertise with regard to real-time system design and debug issues. Its iterative, semi-automated approach facilitates better use of the experts' time, and therefore is a productivity enhancement tool. When the personnel resources are better used, the job of maintaining such complex systems is less expensive and less risky.

CARE has other benefits to the maintainer, including:

- Independent Verification & Validation (IV&V) of entire systems
- Validation and creation of documentation
- Identification of additional entries into the system for security assessment
- Evaluation of system performance

There are many other intangible benefits that are obtained when a system is properly documented and maintained. The benefits of this approach should be considered from the user's perspective as well. Systems that operate properly help maintain the confidence the user requires of them as well as help in maintaining morale among the ranks.

References

[1] G. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," Psychological Review, Vol. 63, 1956.

[2] C. Jones, "Applied software measurement: The software industry starts to mature," American Programmer, June 1991.

[3] M. Halstead, Elements of Software Science, New York: Elsevier, 1977.

[4] T. McCabe, Cyclomatic Complexity, National Bureau of Standards, Special Publication, 500-99.

[5] MIL-STD-490A, Specification Practices, US DoD Standard, June 1985.

[6] MIL-STD-2167A, Defense System Software Development, US DoD Standard, February 1988.

[7] D. Parnas, "On the criteria to be used in decomposing systems into modules," Communications of the ACM, Vol. 5, No. 12, pp. 1053-1058, December 1972.

[8] E. Yourdon, "Relativity of Real-Time Systems, Part 4: Examples of real-Time System Failures," Modern Data, April 1972, pp. 52-57.

[9] J.C. Dickson, J.L. Hesse, A.C. Kientx and M.L. Shooman, "Quantitative Analysis of Software Reliability, Proceedings of 1972 Annual Reliability and Maintainability Symposium, Institute of Electrical and Electronics Engineers, IEEE Cat. No. 72CHO577-7R. New York: 1972, pp. 148-157.

[10] Frederick P. Brooks, Jr., The Mythical Man-Month (Reading, Mass.: Addison-Wesley, 1975).

AUTOVAL (AUTOMATED VALIDATION) ENHANCEMENTS

Steven A. Walters
Science Applications International Corporation
Dayton, Ohio 45432

Abstract

The Operational Flight Programs (OFPs) that run in Embedded Computer Systems (ECSs) in modern aircraft must undergo extensive and rigorous validation testing before they are released for flight. Similar validation testing is required for the highest fidelity flight training simulators to assure that they behave like the actual aircraft. Until now, this validation testing has been a very costly, labor intensive, manual process. The Avionics Logistics Branch of Wright Laboratory (WL/AAAF) and SAIC have just released Version 2.11u of AutoVal (Automated Validation) that completely automates the actions of test engineers for real-time testing of OFP and flight simulator software. A follow-on program is now underway to implement several new features for AutoVal, including language sensitive editing, automatic learning, and automatic verification of visual symbols on aircraft displays. We are extending AutoVal's X/Motif-based graphical user interface to incorporate a language sensitive editor and on-line help to increase the ease and efficiency of developing AutoVal command files. The AutoVal *Learn Mode* will also speed the development of command files by monitoring a test engineer's actions while manually performing a test and automatically creating a command file to reproduce those actions. Finally, pattern recognition technology will be integrated with AutoVal to enable AutoVal to verify data and symbology for Head-Up Displays (HUD) and other cockpit displays that currently require human visual verification. This paper describes the details of these new AutoVal enhancements and how they may be used to further reduce the cost and increase the effectiveness of testing avionics software in modern aircraft.

Background

We are seeing a virtual explosion in the use of computers in aircraft. As the size, cost, and power consumption of computer hardware decreases, aircraft and avionics design engineers are using them, and the software they contain, in creative new ways to push the capabilities of new weapon systems to dramatic new heights. Computers and their software are also playing a major role as designers retrofit existing aircraft to expand the capabilities and extend the life of earlier generation, high performance airframes. Not only are more and more computers being used, but the performance and memory size of these computers are also rapidly increasing. This is producing a corresponding increase in the size and complexity of the embedded software in these computers. Growing numbers of computers combined with the increase of software size in each

computer has produced exponential growth in the total quantity of embedded software in emerging weapon systems.

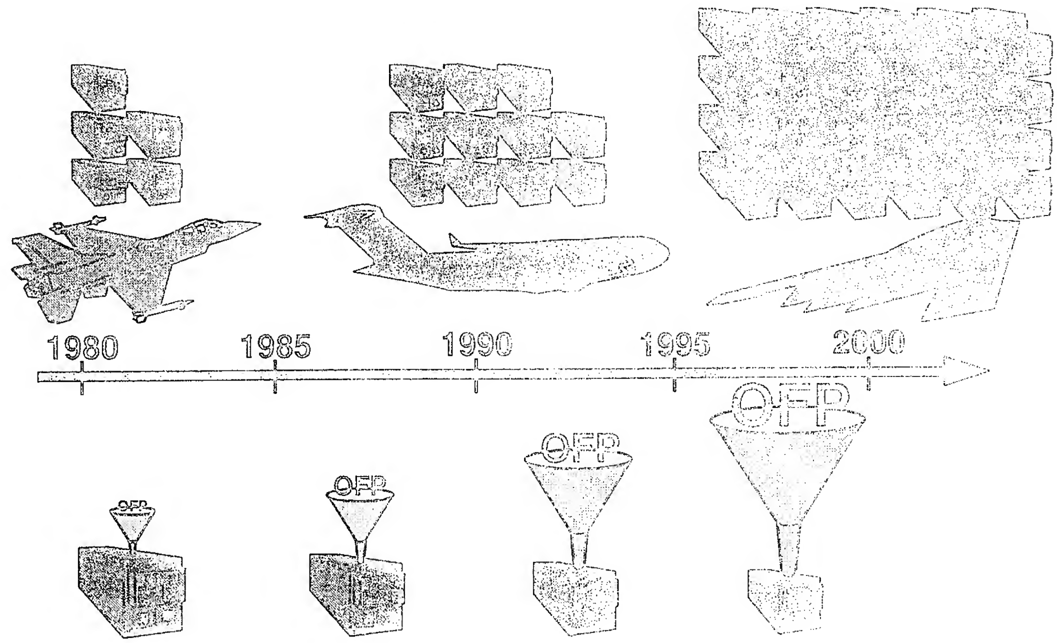


Figure 1, Avionics Software Trends

The tremendous growth in the amount of embedded software in aircraft currently under development will have enormous consequences for the government in terms of future software maintenance and testing. Because the embedded software in a modern weapon system controls very expensive and very lethal equipment, it must be thoroughly and rigorously tested before it is released for flight. Over the next five years, the government's capacity for testing embedded software will have to expand by an order of magnitude to support the new advanced aircraft that will be released into the inventory by the end of the century.

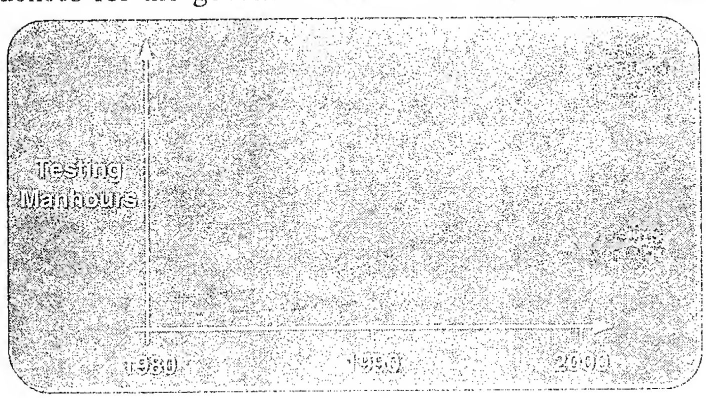


Figure 2, Avionics Software Testing

SAIC is working with WL/AAAF to help the government increase testing capacity by developing reusable tools and technologies that can be used by government testing organizations to automate the test process. One of these tools is called AutoVal, which stands for "automated validation." AutoVal was first released by WL/AAAF four years ago and is currently in use at the Ogden Air Logistics Center (OO-ALC) for F-16A/B OFP testing. By reducing the time required for testing an OFP, automated testing increases the amount of software an individual test engineer can productively handle. It relieves test engineers of having to sit at a test station performing rote tests and allows them to focus on the important task of designing more thorough and better

quality test procedures. This results in not only better tests, but also tests that are performed more accurately and repeatably. Automated testing also significantly reduces the time required to field OFP changes, thus increasing readiness. Performance studies with AutoVal using actual OFP test cases have demonstrated up to a 100-fold reduction in testing time over manual testing. As OFP quantity and complexity grow, automated testing may be the only way to cost effectively keep up.

AutoVal Overview

The AutoVal concept is very straightforward. A test engineer converts the Formal Qualification Test (FQT) to a series of AutoVal commands. The AutoVal command language has been carefully designed to be intuitive and English-like, and to closely match the terminology and structure of typical FQT test steps. Commands are provided for the complete test process, including both the actions or test stimuli, as well as the monitoring and verification of test results. After a test has been translated to a series of AutoVal commands, the commands are then stored in a file. When installed on an avionics test station, AutoVal can read these commands and control the operation of the test station to conduct the test. There are a number of "automated" tools, some called "scenario generators", in use at various support facilities. The big difference between AutoVal and these others is that AutoVal provides both the test stimulus and verification of the results -- it can even generate the test report, automatically.

AutoVal was designed from the beginning to maximize portability and reuse. It is based entirely on open standards. All of the software is written in Ada with X/Motif for the user interface, and the current version is hosted on the UNIX operating system. The interface to the test station is defined through data files to minimize the impact both to AutoVal and to the test station software when installing it on a system. AutoVal also adapts easily to the specific testing requirements of any avionics system.

Figure 3 shows how AutoVal interacts with an avionics test station. AutoVal replaces the control inputs that are produced when a test engineer pushes buttons on an avionics control panel or when he "flies" the simulation. AutoVal can also replace or supplement inputs from the simulation or from other avionics subsystems for error injection or failure mode simulation. In effect, it can do, automatically, whatever a test engineer does, manually, to provide test stimuli. AutoVal then intercepts data coming from the

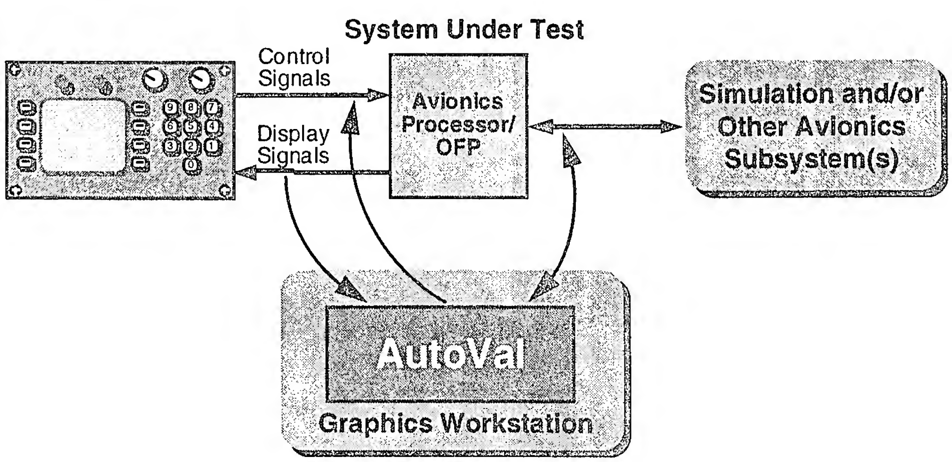


Figure 3, AutoVal Control and Monitoring

embedded computer to verify the correct results.

AutoVal's Command Language

The AutoVal Command Language has four major component subsets. They are:

- Native Commands
- Macro Commands
- Switch Definitions
- Symbols

WL/AAAF collaborated with OO-ALC during the original development of the AutoVal and used an F-16A/B test station for the original concept demonstration. We used several F-16 FQTs as background information for devising the original command language, and then tested the viability of the language by translating different sections of the latest F-16A/B FQT into AutoVal command files. As we periodically encountered test steps that weren't supported by the language, we would create new commands to accommodate them. After creating more than a hundred new commands, we realized that we no longer had a general purpose automated testing language, but a very focused F-16A/B fire control computer OFP testing language with little application to other aircraft, or even other computers within the F-16A/B. At that point, we went through the command language and identified those commands that were truly generic and application independent. These became the AutoVal Native Command Language.

The native commands are generic, English-like commands that provide basic functions for test case creation, such as memory access, arithmetic/logic operations, flow control, file handling, etc. They were purposely structured to be completely independent of any specific test application domain. As we converted several test cases to the native commands, we found that there was so little connection to the application that the test cases became very cryptic and difficult to relate to the original FQT. We needed application-specific commands, but we didn't want to have to create different versions of AutoVal for each different test station. This spawned the creation of the AutoVal Macro Command Language.

With macro commands, a test engineer can create application-specific commands tailored to whatever unique types of test steps are required without affecting the native command language. Macros can have up to nine arguments and are built using native commands and/or other macros. Once created, a macro command is used in the same way as a native command to build test cases. Macros are one of AutoVal's most powerful features. They can be used to encapsulate very complex test functionality that isolates the inner workings of the test station simulation from the test engineer and makes it faster and easier to get test cases up and running. Macros also allow AutoVal to be adapted to testing across a broad range of systems and applications.

The most common test action in an FQT is the activation of a switch on a real or simulated avionics control panel by the test engineer. Most test cases are lengthy sequences of knobs turned, buttons pressed, switches toggled, etc. followed by verification of a state change in the avionics. To capture this construct, AutoVal incorporates Switch Definitions that describe the

switches needed for a test case. These switch definitions name the switches and all of their possible states. They also describe the behavior of the switches, e.g., momentary/toggle/rotary, state sequence, timing, etc., and define the data locations and patterns of the switch output data. Switch definitions are stored in files so that they can be dynamically invoked at run-time. This use of abstraction and dynamic binding has the benefit that when changes are made to a switch's output data or behavior due to updates or modifications to the test station, these changes don't require modification of the test cases that use that switch. Figure 4 shows a representative switch definition.

```
-- Define Landing Gear Switch
-- with positions UP/DOWN
Begin_Switch Landing_Gear
UP:   Write Landing_Gear_Discrete = 1
      Wait 0.5
DOWN: Write Landing_Gear_Discrete = 0
      Wait 0.5
End_Switch
```

Figure 4, Switch Example

Symbols represent test station simulation variables or other data in the system to which AutoVal has access during execution of a test case. This allows test engineers to reference data using meaningful, symbolic names. By creating symbols and using them to build test cases, the test engineer makes the test cases more useful and understandable. Also, when modifications are made to the test station that change the values associated with one or more symbols, only the symbol file needs to be updated. The test cases that use those symbols remain unchanged. Figure 5 presents an excerpt from a typical symbol file.

```
Symbol ON                        1
Symbol OFF                       0
Symbol Landing_Gear_Discrete  0FDC
Symbol Hold_Vertical_G        2193
Symbol Hold_Altitude          2194
Symbol Pitch_Attitude         00EF
Symbol Altitude               00F0
Symbol Mach_Airspeed          0104
```

Figure 5, Symbol Examples

AutoVal's User Interface

AutoVal's most recent release, Version 2.11u, involved sweeping changes to the user interface. Earlier versions were hosted on Digital Equipment Corporation (DEC) VAX computers under the VMS operating system and used a terminal-based approach implemented with DEC screen management utilities. Because of the rapid replacement of terminals by graphics workstations in most facilities, WL/AAAF tasked SAIC with converting AutoVal's user interface to a windows-based style using X/Motif on a modern UNIX workstation. This conversion produced two important benefits. First, it made AutoVal much more portable. X/Motif has finally emerged as the de facto open standard for user interface design on high performance graphics workstations. AutoVal is now readily portable to any UNIX-based workstation with X/Motif and Ada. The second benefit was that it made AutoVal much easier and more intuitive to use. Since we carefully followed the Open Software Foundation (OSF) Motif Style Guide during the conversion, anyone who is familiar with the Motif windows environment can easily and productively navigate through all of the features and operations of AutoVal with little or no reference to the user's manual.

Figure 6 shows all of the elements of the new AutoVal X/Motif user interface. The main AutoVal menu provides selections for invoking all of the AutoVal control functions, as well as file manipulation (i.e., Open, Save, Save As, etc.) and editing functions (Cut, Copy, Paste, etc.). In the Edit window, the test engineer can create, edit and display AutoVal test cases. The Log File window scrolls all of the AutoVal messages at the same time they are written to a log file for post test review. The test engineer can use AutoVal to interactively control the test station by typing AutoVal commands in the Interactive Command Line window. At the bottom of the AutoVal main window are buttons that implement the most commonly used controls from the main AutoVal menu.

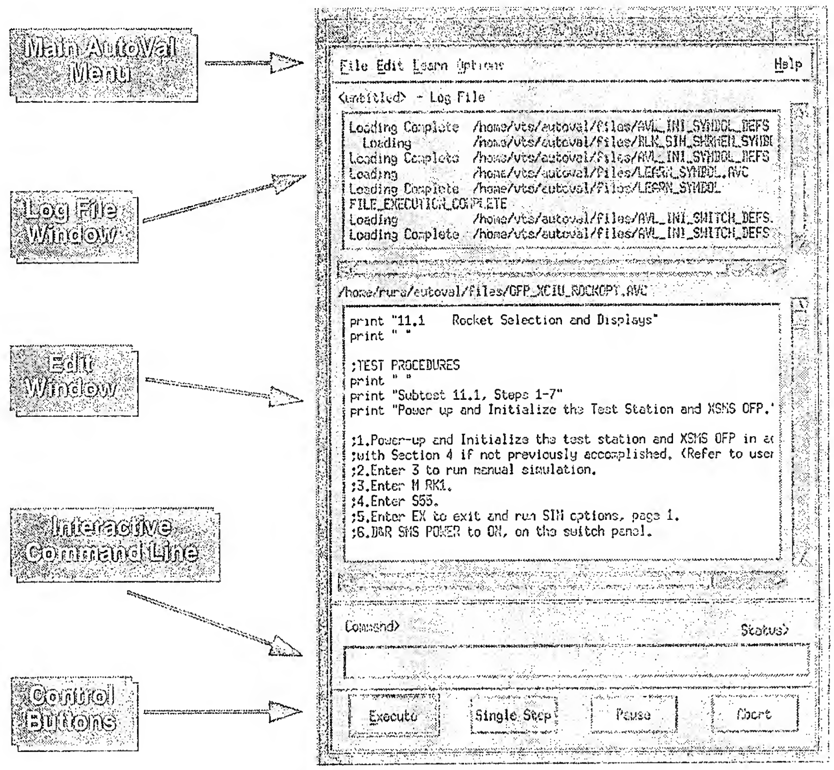


Figure 6, AutoVal User Interface

New Features and Enhancements

SAIC and WL/AAAF are underway with the development of the next two versions of AutoVal, Version 3.00 and Version 4.00, respectively. The goal of AutoVal Version 3.00 is to further increase the efficiency of the test engineer during test case development, by making it faster and easier to build test cases. AutoVal Version 4.00 addresses the only remaining avionics test process that we haven't been able to automate, thus far -- visual verification from a cockpit display.

To accomplish our goal for Version 3.00, we are adding two major new features. The first is an addition to the X/Motif-based user interface called the Command Palette. The Command Palette is an auxiliary window that provides a language sensitive editing capability for creating and modifying AutoVal test cases. The second is called Learn Mode. With Learn Mode, an experienced test engineer can direct AutoVal to monitor his actions as he manually performs a test and automatically convert those actions to equivalent AutoVal commands. When later executed by AutoVal, these commands exactly replicate the actions originally performed by the test engineer.

With Version 4.00, we are implementing AutoVal for Displays. AutoVal for Displays will capture cockpit display images and perform pattern recognition of the symbols and text to directly verify

data and behavior from the display. This will eliminate the need for a test engineer to do post-run verification from display snapshots or to be present during automated testing to visually monitor and verify the displays.

Command Palette

Over the past four years of using AutoVal and observing test engineers working with AutoVal, we have noticed that a lot of time during test case preparation is spent looking at reference materials because of the large quantity of information needed to construct a test case. This starts with the command language, itself. As English-like and intuitive as we have tried to make it, AutoVal native commands and macros still have a specific syntax that must be used to produce test cases that will operate properly. In addition to the 45 AutoVal native commands, test engineers using a typical test station will also define a hundred or more macros to support testing with a particular LRU. Next are the switch definitions. The avionics control panels used for testing contain dozens of switches, each with from two to ten positions. When a test engineer reaches for a switch during manual testing, he doesn't have to remember the exact nomenclature of the switch and position he wants to activate. When writing an AutoVal test case to perform that same action, he does. Finally, we have symbol definitions. A typical test station uses more than a thousand symbols for data within the system. Hundreds of these are needed to prepare the test cases used for testing a complete OFP. While preparing a test case, the test engineer is constantly referencing the syntax, arguments, spelling, and other details about the commands, macros, switches, and symbols he is using to build the test case.

The purpose of the Command Palette is to streamline this reference process and integrate it with test case editing to improve the test engineer's efficiency and to reduce the time required to build operational test cases. As shown in Figure 7, the Command Palette window has four sub-windows, one each for AutoVal native commands, macros, switches, and symbols, respectively. The Command Palette is invoked from the main menu on the AutoVal window and appears adjacent to the AutoVal window on the test engineer's workstation screen. The Command Palette supports both a reference function, including an extensive on-line Help capability, and a language sensitive editing function in conjunction with AutoVal's Edit window.

In each of the Command Palette sub-windows, the test engineer can scroll through the list of items in that sub-window. When an item is selected (pointed to with the cursor and "clicked on"), the complete syntax for that item is displayed. For commands and macros, this consists of the command or macro name and all of its

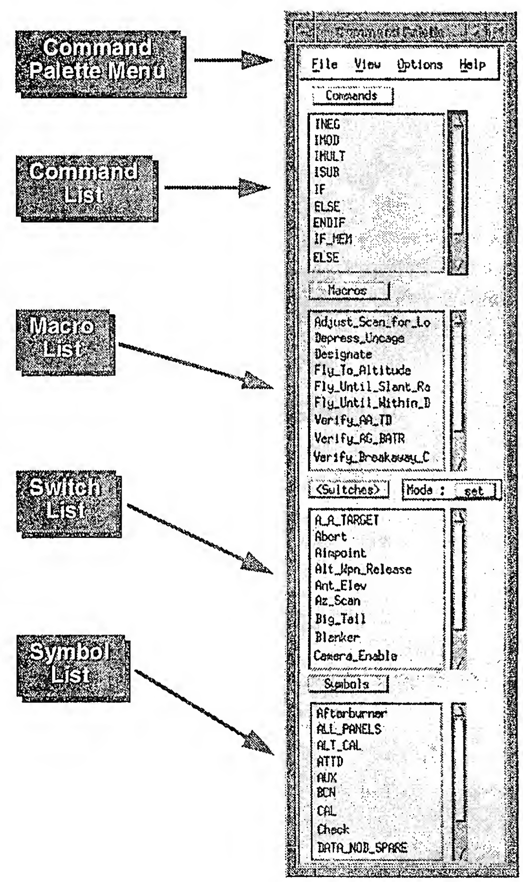


Figure 7, Command Palette

associated arguments. For switches, this consists of a list of all the possible positions for the selected switch. For symbols, the syntax is simply the symbol name. After an item is selected, the test engineer can then copy the item or the item with its complete syntax description to the current pointer location in the AutoVal Edit window. The test engineer also has the option of bringing up an expanded text description for the selected item by selecting Help from the Command Palette menu.

With AutoVal it is possible to load and have simultaneously available a number of different files for macros, switches, and symbols. With the Command Palette, we give the test engineer the option of which of these files he would like to have displayed in the respective Command Palette sub-windows. The test engineer starts the selection by pressing the button above the desired sub-window. This action displays a list of the loaded files available for that sub-window. The test engineer can then select a particular filename or "All". This feature is useful for the test engineer who is working, for example, on the radar portion of a large OFP test case and doesn't want to scroll through the dozens of other macros loaded for that test case just to find the radar-related macros.

Learn Mode

Learn Mode is another new feature designed to speed up the process of creating command files. It is based on the premise that an experienced test engineer can sit down at the test station and perform a test faster than he can type the commands for that test into AutoVal. As shown in Figure 8, Learn Mode has two monitoring processes that "watch" what the test engineer is doing on the system. These monitor tasks relay control state change data to a task that uses switch and symbol tables to translate the state change data into command arguments that are then used to generate AutoVal command. When played back through AutoVal, these generated commands will replicate the actions performed by the test engineer.

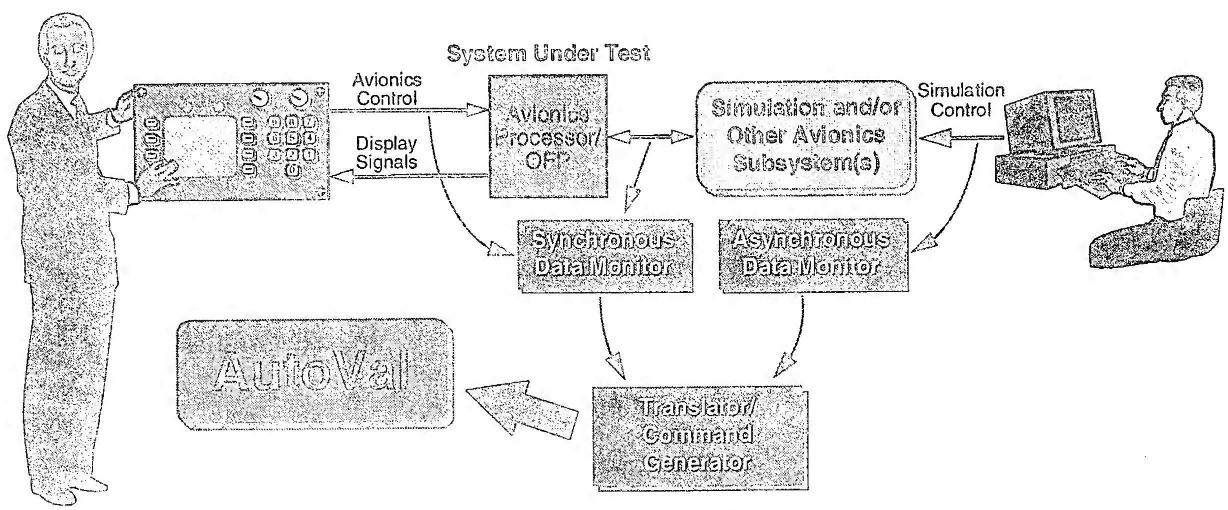


Figure 8, AutoVal Learn Mode

Obviously, Learn Mode can only be used to generate the action/stimulus commands for an AutoVal test case. The test engineer still has to manually insert validation commands at

appropriate points in the test case. (We haven't figured out how to monitor the test engineer's mind, yet, to automatically generate the validation commands.) The test engineer can do this during a Learn Mode session by pausing the system and typing a validation command at the Interactive Command Line window. This command will be inserted after the commands that were generated prior to the pause. The test engineer then resumes Learn Mode to continue automatic command generation. As an alternative, the test engineer can proceed through an entire test case and automatically generate all of the action or stimulus commands. After completing the entire sequence, he can then use the AutoVal editor (and Command Palette) to scroll through the generated test case and insert all of the validation commands at one time.

AutoVal for Displays

As cockpit displays get "smarter" and display OFPs become more prevalent, full automated testing requires that AutoVal be able to "see" the display to determine whether certain types of tests have passed or failed. In the past, this wasn't necessary because AutoVal could intercept the MIL-STD-1553 messages going into the display device and test the message content for correctness. The assumption was that, barring an electrical failure in the display, a correct input message always produced the correct image. With processing now taking place inside the display system, checking the messages going in does not provide absolute assurance that the proper images will actually be produced.

Our strategy (Figure 9) is to attach to the video signal line between the display generator and the display and send the signal into a video scan converter. The scan converter will convert the stroke and/or raster signals created by the display generator into a standard video format. This standardized signal will then go to a video frame grabber that will produce a bitmap representation of the display image. At this point, the problem separates into two issues, character recognition and dynamic symbol analysis, which we are handling with different approaches.

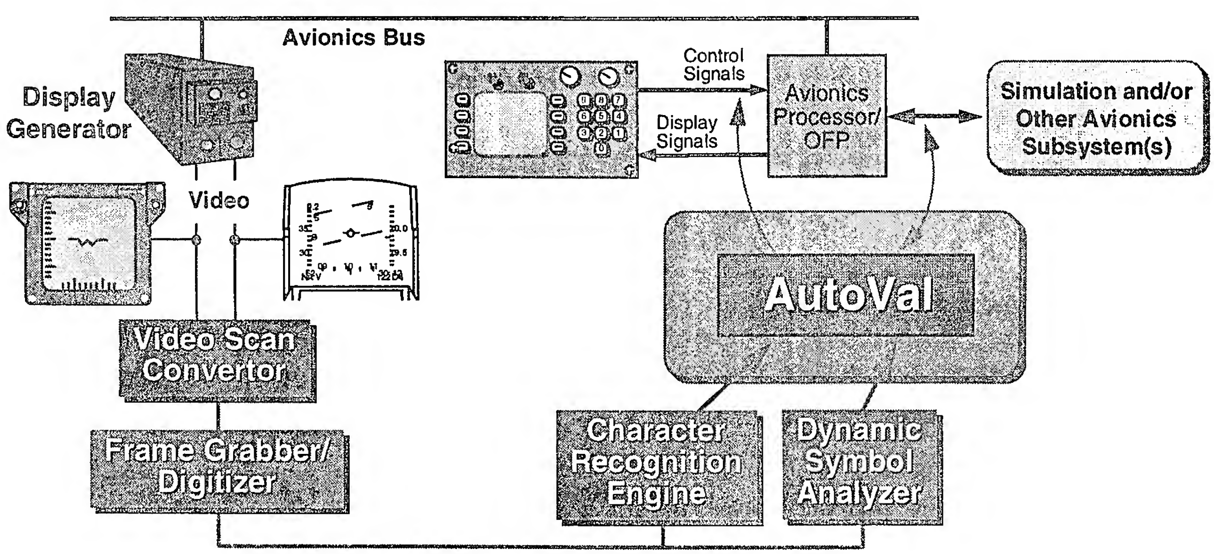


Figure 9, AutoVal for Displays

A large amount of the display data that AutoVal needs to test is in the form of numbers, alphabetic characters, or static symbols that are displayed in fixed fields within the display image. The process of converting the contents of these fields to ASCII text strings is relatively straightforward and will be handled by a COTS character recognition product.

Analysis of the behavior of dynamic symbols is a more challenging issue. A number of symbols on HUD and radar displays change their location, orientation, and/or geometry as a function of various conditions and states within the avionics system. Whether a particular test passes or fails depends on symbol characteristics such as where the symbol is, what direction it is moving, how fast it is moving, what its juxtaposition is to other symbols, or what its shape is. To be consistent with AutoVal's current architecture, this analysis also has to be done in real-time. We are designing a separate Dynamic Symbol Analyzer task to take the bitmap data from the frame grabber and determine a pass/fail result against predefined symbol behavior criteria.

Conclusion

Automated testing of embedded OFP software will become increasing crucial as we approach the end of the Twentieth Century due to the tremendous amount of software that will require support. AutoVal Version 2.11u is a mature, reusable product that is available to meet this challenge. AutoVal automatically performs both test stimulus and verification of results. It has demonstrated dramatic reductions in OFP testing time and increases in test accuracy. It was designed for reuse from the beginning and is host computer and test station independent. AutoVal is written entirely in Ada with X/Motif for the user interface. It has a straightforward native command language and is easily adapted to new systems and applications through the creation of macro, symbol, and switch definitions for the new system.

AutoVal is actively supported by WL/AAAF and SAIC, and a number of new upgrades and enhancements are underway. AutoVal Version 3.00 will be released in August 1995 and will include the Command Palette and Learn Mode to speed up the process of test case preparation. AutoVal Version 4.00 will follow in September 1996 and will incorporate real-time display pattern recognition into AutoVal's verification function. These new releases will further extend AutoVal's automated testing capabilities to encompass the testing needs of aircraft entering the inventory through the end of the century.

APPENDIX to the TFWGCON '95 PROCEEDINGS

CONTENTS

ABSTRACT

HIGH VOLUME DIGITAL DATA RECORDERS FOR MIL STD 1553 AND PCM TELEMETRY APPLICATIONS

DESIGN ANALYSIS ASSOCIATES INC .

APRIL 12, 1995

Data recording requirements in the Aerospace and Military markets are many and varied in both purpose and objectives. The type of equipment currently available to the user is also quite diverse. The digital recording equipment to be described in this paper was designed to be both multipurpose in nature and a very cost effective solution in most applications where PCM or 1553 data needed to be recorded in high volume. Design Analysis Associates (DAA), also considered the following factors in the design of the recorders described in this paper.

1.Easy of use
2. Off-the-shelf software support for data analysis
3.Ablility to support lab and flight applications
4.Convenient size and weight
5.Ease of ability to upgrade both software and hardware improvements

The 3100 series digital recorders offered by DAA incorporated all of these factors into its design. The 3100 series recorder is specifically used to record data from the Mil-Std-1553 bus.It has the capability to record two dual redundant 1553 buses at one hundred percent bus loading for a period of two hours. The data storage capacity of the 8mm magnetic tape media is five gigabytes uncompressed. The 8mm tape cassettes are both inexpensive and easy to store when compared to the larger tape reels used in the older analog recorders. The 3100 series are available in both a desktop configuration for simulation labs or ground stations and a ruggedized version for in-flight testing . The ruggedized unit also lends itself to use in heavy armored vehicles such as tanks, personnel carriers, or landing craft where the 1553 bus is utilized.

The 4100 series digital recorders offered by DAA have been designed to record PCM telemetry data in a digital format on the same type of 8mm magnetic tape previously mentioned in the description of the 1553 recorder. One of the major advantages of the DAA telemetry recorder is that the system decommutates the PCM stream and records the decommutated words on tape in a digital format. When all of the required data has been recorded, the tape can be removed from the recorder and analyzed without the need to reconstruct the PCM stream using extensive ground support equipment . One can see the advantages of being able to analyze data in the field using a general purpose computer and not having the possible delays of having to wait for access to ground support equipment. The PCM recorder can record two PCM streams in word sizes from 8 bits to 32 bits. The

ABSTRACT

recorder also has a serial IRIG-B input for time stamping and a 1 millisecond internal time stamp if IRIG-B time stamping is not present.

The paper presented will describe both recorders in more detail and show their advantages over some of the competing technologies.

HIGH VOLUME DIGITAL DATA RECORDERS

With the ever increasing complexity in the flight control systems and avionics associated with both commercial and military aircraft, the need to collect more meaningful data with greater accuracy and reliability continues to grow in importance. The ability to collect large quantities of data and analyze the data in a cost effective manner became a major factor to be considered when selecting a system for both ground station testing and flight testing. While being very effective for many ground based activities , the large reel tape recorders that have been used for years in the aerospace industry are not suitable for flight applications where space and weight are at a premium. Often, reconstructing the data stream with other ground support equipment is required prior to data analysis. The delays involved between collection and analysis can be costly both in terms of solving a major problem and the time and expense of the corrections required.

Design Analysis Associates,Inc., (DAA) became involved with the collection of 1553 data and the analysis of that data in laboratory simulations for various applications at the China Lake Naval Air Warfare Center in the early eighties. To facilitate longer recording times and ease of use , DAA set out to design digital data recorders that could be used for ground based testing and also provide a ruggedized system that could be used in an in-flight test situation. The criteria used to come up with the design concept was as follows:

1. Ease of use
2. Off-the-shelf software support for data analysis
3. Ability to support both lab and flight test conditions
4. Convenient size and weight
5. Ease of ability to upgrade software and hardware

The increasing use of the Mil-Std -1553 data bus prompted the design of the 3100 series digital data recorders that had the capability of recording data from two dual redundant 1553 buses at full bandwidth for two hours. This design was facilitated by the availability of the Exabyte 8505 8mm tape drive. This 8mm magnetic tape was a very convenient media for data storage from both a size and cost standpoint. After data had been recorded , the tape could be easily removed from the recorder and analyzed. This will be discussed in more length later in this paper. In order to optimize the recording time available, a buffer was incorporated into the design to allow the tape drive to idle when there was no data being transmitted or data rates were very slow . When the designated amount of data had filled the buffer, the tape drive would come up to speed and the data would be transferred to the tape. This feature keeps the tape from continuously streaming and allows the optimum data density of the tape to be fully utilized.
Therefore, if the bus loading is less than 100%, or only one dual redundant bus is being utilized, the recording time is extended beyond the two hour minimum. For example, if both dual redundant buses have 25% loading , the recording time available would be eight hours. Since most flight tests are less than four hours in duration, a single tape would probably suffice. Should the test time required exceed the data storage capacity of a single tape, the design of the recorder is such that it is very easy to remove the filled tape and insert an unused one provided that access to the recorder is available to the flight test crew.

One of the criteria for the design of DAA's digital recorders was ease of use. Essentially, once the recorder has been connected to the power source and the connections to the data source have been made, operation of the system form either a keyboard or remote switch control box is extremely easy. The commands from either consist of record, pause, stop, and rewind. Menu driven operating software has been provided by DAA that allows the user to view raw data, transfer data to disc memory, configure the system to record specific remote terminals and also position the tape to extract a specific segment of data. In the case of the ruggedized unit for flight testing, the transfer function is not available. However, pins on the connectors provided do allow a keyboard and monitor to be `utilized with the unit to take advantage of the other features of the design.

An earlier mention was made in reference to the analysis of data collected . DAA has had an informal working relationship with a software company, BBN, for over five years. BBN has developed software that provides filtering for the data recorded by DAA's system and also has a telemetry and general purpose software package that can analyze 1553 and PCM data . Once a data dictionary has been created in conjunction with the user, a tape to be analyzed can be placed in a backup drive , read, and the data configured as the user desires. The major advantage of this scenario is that the recorder and software combination provides a cost effective, off-the-shelf solution to data recording and analysis. The user does not have to write any code, all of the work has been done. If the user already has software that fills their requirements, it too can be used effectively with the DAA recorders.

There are several differences in the lab and flight versions of the recorders. The lab version has an Ethernet card in the system to allow the user to make the data available to their network disc. One other difference between the two models is that the lab version has a hard disc not available in the flight model. This is used for the transfer function and also to record to hard disc if required.While magnetic tape has been the media offered with the design being described here, if a customer would prefer a hard drive memory, there are high density disc drives that could be supplied as an alternative to the tape system.

The ruggedized model of the data recorder is supplied with a shock mount system . The nature of the system depends on the type of aircraft the system will be flown on and the shock and vibration levels it will be exposed to. The ruggedized model can be furnished with a battery backup option that will allow the recorder to operate for 15 to 20 minutes in the event of a power failure. It also prevents the tape drive from rewinding if a momentary power glitch occurs during the switch over from auxiliary power to aircraft power. An extended temperature version is also offered that provides resistive heating if required and thermoelectric cooling where operating temperatures may exceed the normal levels that the system operates under. The unit is O-ring sealed to prevent moisture and dust from causing reliability problems. An internal fan circultes the air inside the recorder to provide a more uniform internal temperature. The case is of aluminum construction and the small size accomodates most spaces available on aircraft for recording equipment.

Provisions for time stamping have been made to allow the user to correlate other events or data from range testing or other sources. An input for IRIG-B time coding is provided and the time stamp is incorporated into the data stream. In the absence of an IRIG-B time code source, the recorders provide a one millisecond time stamp that can be used for locating specific data on the tape and data transfer.

The information provided to this point has been directed towards the 1553 recording capabilities of DAA's systems, but many of the features are common to the other data recorders that are available from DAA . The 4100 series recorders were designed specifically for PCM recording applications. The design approach was again to make the unit easy to use while providing some major advantages over existing technologies. As the PCM data stream comes into the recorder, it is decommutated and put onto tape. As in the case of the 1553 recorder, IRIG-B time can be imbedded in the data stream or the internal time stamp can be utilized. The recorder series can accept two PCM channels ranging from 8 to 32 bit words . The recording time available is two hours with a data transfer rate of 400 kilobytes per second or one 32 bit word every 10 microseconds. Unlike many of the older systems used for recording PCM data, the DAA approach permits the user to remove the tape with the recorded data from the machine and go directly to data analysis without the need to spend additional time and money going back through expensive ground support systems to reconstruct the PCM stream and transfer that data to tape.
With the BBN software, the recorded data can be analyzed immediately at the test site or any convenient location with the backup drive and computer capability needed.

Many times data recording requirements are not met by the off-the-shelf designs. Recognizing this fact, DAA came up with the flexibility in their data recorders to be able to provide some degree of customization with out having to go to costly redesigns at unaffordable pricing. Several examples of what could be made available in a timely fashion at attractive pricing are as follows:

1. A 1553 recorder with four dual redundant bus capability
2. A recorder having one 1553 input and one PCM input
3. Recorders that use media other than 8mm tape
4. Recorders with custom card configurations such as AtoD Converters, GPS Receivers, and Video

Since the recorders collect all of the data on the buses monitored unless the record configuration has been set to look at specific RT's, their applications can include system test and verification, troubleshooting of new avionic upgrades, finding intermittent failures or problems, certification of systems on the buses monitored, and general data collection for various flight tests. While all of the information to this point has been directed towards aircraft applications, the data recorders can be used for vehicular applications such as tanks , armored personnel carriers, missiles, landing craft, and shipboard applications where any data needs to be recorded from a 1553 bus , a PCM stream, a 485 utility bus , and many analog or digital sources.

The attachments to this paper are the general specifications for the 3100 series recorders, the 4100 series rcorders, and the 485 utility bus recorder which has not been mentioned in great detail but is quite similar to the 1553 bus recorders. To briefly summarize what has been presented, it can be stated that the digital recorders from DAA provide the user with an accurate and reliable off-the-shelf approach to cost effective data collection and analysis. For more information on the products described, or for an equipment demonstration , Please contact either Joe Corbett or Tom Elsby at Design Analysis Associates Inc.

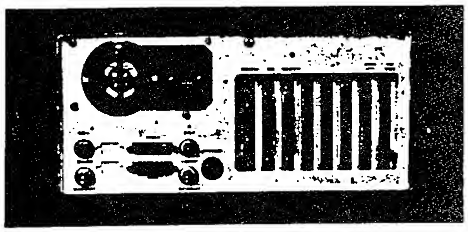
Model 3100-DT Mil-Std-1553 Digital Tape Recorder

The Model 3100-DT is used to log selected or complete traffic from two dual redundant Mil-Std-1553 buses for a period of two hours and more. It will log command, data, and status words, along with additional information associated with each 1553 word, as they are transmitted over the bus. It logs a command/data bit, a manchester error bit, and bus gap formation for each word on the two buses.

The Model 3100-DT does not require the tape drive to be kept streaming. This means that a 1553 bus usage of 50% would allow the user to record data for 4 hours.

8mm Digital Tape Storage

The Model 3100-DT system uses inexpensive 8mm video tapes for high volume digital tape storage. The tape containing the logged data is written by the new EXABYTE 8500 tape drive in an ANSI standard file and is compatible with the tape backup devices sold by many vendors for disk backup purposes.



The tape is created with a time based directory which allows the data on the tape to be rapidly retrieved using the new tape positioning features of the 8500 tape drive. The user is able to access any portion of the data on the tape at an average access time of 1.5 minutes.

IRIG-B Time Stamping of the Data

The Model 3100-DT will log time code events from an external IRIG-B time code generator. The time stamp information can be used to correlate the data on the tape with external events or data. The Model 3100-DT system also has a 1 millisecond time "tick" generator, which can be used for elapsed timing.

Logs User Discretes

The Model 3100-DT allows the user to connect up to 28 digital discrete lines, which are sampled along with the time information. These discrete lines can be used to monitor switch settings, analog readings from an A/D converter, or other user data.

Operation

The logging process can be activiated by the keyboard or the user can control the unit remotely with a RECORD switch and a REWIND switch. After loading the tape the user can start the logging process by activating the RECORD switch. By deactivating the RECORD switch the user can pause the recording. The recording process will stop when the user activates the REWIND switch.

The user can select which RT's will be monitored for logging by invoking the configuration software provided with the Model 3100-DT and following the menu prompts.

Analysis of the Data

The data can be transferred to another computer via the provided Ethernet interface for analysis. For more information about commercial software analysis programs, contact Design Analysis Associates or your local representative.

Ordering Information

HV - 3100 - DT, RM, RG

DT - Desk Top
RM - Rack Mount
RG - Ruggedized

For more information telephone Design Analysis Associates:
Tel: 801-753-2212
FAX: 801-753-7669

pecifications

Dimensions
- Height: 7 inches
- Width: 24 inches
- Depth: 24 inches

Weight
- Net: 35 lbs.
- Shipping: 40 lbs.

Power
- 120 VAC +/- 10%
- 70 Watts

Environment
- Temperature
  - Operating: 41 d F to 104 d F
  - Storage: -4 d F to 185 d F
- Humidity
  - 20% to 90% Non Condensing

Mounting
- Table Top
- Rack Mount

Interface
- Two Mil-Std-1553 A/B (Each Dual Redundant)
- IRIG-B Parallel BCD Output Time Code Generator
- Custom User Discrete

Logging Media
- Standard 8 mm Video Tape
- Cartridge (3.7 x 2.5 x 0.6 ins.)
- Formatted Capacity

Tape Size	Length (Ft.)	Maximum Formatted Capacity in Mega Bytes
256	45	582
512	90	1166
1024	180	2332
1563	540	3500
2048	360	4664

Transfer Rate
- 492 KBytes per second sustained
  - (Full bandwidth of 1553 data, plus time stamps and digital discretes)

Reliability
- Non-recoverable error rate:
  - Less than one in 10**13 bits read
- MTBF: > 20,000 hours at typical usage

Preventative Maintenance
- Periodic head cleaning

Your Local Representative Is:

DESIGN ANALYSIS ASSOCIATES INC.
75 West 100 South
Logan, Utah 84321
(801) 753-2212
FAX (801) 753-7669

Model 3100-RG Mil-Std-1553 Ruggedized Digital Tape Recorder

The Model 3100-RG is used to log selected or complete traffic from two dual redundant Mil-Std-1553 buses for a period of two hours and more. It will log command, data, and status words, along with additional information associated with each 1553 word, as they are transmitted over the bus. It logs a command/data bit, a manchester error bit, and bus gap nformation for each word on the two buses.

The Model 3100-RG does not require the tape drive to be kept streaming. This means that a 1553 bus usage of 50% would allow the user to record data for 4 hours.

8mm Digital Tape Storage

The Model 3100-RG system uses inexpensive 8mm video tapes for high volume digital tape storage. The tape containing the logged data is written by the new EXABYTE 8500 tape drive in an ANSI standard file and is compatible with the tape backup devices sold by many vendors for disk backup purposes.

IRIG-B Time Stamping of the Data

The Model 3100-RG will log time code events from an external IRIG-B time code generator. The time stamp information can be used to correlate the data on the tape with external events or data. The Model 3100-RG system also has a 1 millisecond time "tick" generator, which can be used for elapsed timing.

Logs User Discretes

The Model 3100-RG allows the user to connect up to 28 digital discrete lines, which are sampled along with the time information. These discrete lines can be used to monitor switch settings, analog readings from an A/D converter, or other user data.

Operation

The logging process is controlled with a RECORD switch and a REWIND switch. After loading the tape the user can start the logging process by activating the RECORD switch. By deactivating the RECORD switch the user can pause the recording. The recording process will stop when the user activates the REWIND switch.

The user can select which RT's will be monitored for logging by invoking the configuration software provided with the Model 3100-RG and following the menu prompts.

Analysis of the Data

The tape containing the data can be read by computers using industry standard 8mm tape drive units which use the EXABYTE 8500 tape drive. For more information about commercial software analysis programs, contact Design Analysis Associates or your local representative.

Current Customers

The ruggedized HVLOG units are currently be used in high vibration environments such as Boeing's CH-47 helicopter and McDonnell Douglas' Apache helicopter.

Ordering Information

HV-3100-RG

For more information telephone Design Analysis Associates
Tel: 801-753-2212
FAX: 801-753-7669

Specifications

Dimensions
- Enclosure:
  - Height: 6.75 inches
  - Width: 10.19 inches
  - Depth: 15.38 inches
- Enclosure and Mount:
  - Height: 8.59 inches
  - Width: 11.69 inches
  - Depth: 19.88 inches

Weight (Including Mount)
- Net: 22.4 lbs.
- Shipping: 32 lbs.

Power
- 16 to 36 Volts DC
- 70 Watts

Environment
- Temperature
  - Operating: 5 deg. C to 40 deg. C
  - Storage: -20 deg. C to 85 deg. C

Vibration (At the Enclosure)

0.3 g rms random vibration spectrum as defined:

Frequency	Slope (dB/Oct)	PSD (g2/Hz)
5-350	0	0.0002
350-500	-6	—
500	—	0.0001

Shock (At the Enclosure)
(1) 3g for 5ms
(2) 2g for 11ms
(3) 1g for 20ms

Note: The above information is the specification at the enclosure if the user wishes to implement his own shock and vibration control. The shock mounts provided with the unit limit the shock and vibration to the enclosure. We have run vibration tests with the unit when mounted on the shock mounts. The test data is available upon request.

Mounting Base
- Barry Controls AOLA SERIES (Military Standard Number MS91405-B1C)

Humidity
- 20% to 90% Non Condensing

Logging Media
- Standard 8mm High Quality Video Tape Cartridge (3.7 X 2.5 X 0.6 ins)

Reliability
- Non-recoverable error rate:
  - Less than one in 10**13 bits read
- MTBF: > 20,000 hours at typical usage

Preventative Maintenance
- Periodic head cleaning

Your Local Representative Is:

Design Analysis Associates
75 West 100 South
Logan, Utah 84321
(801) 753-2212
FAX (801) 753-7669

MODEL 4100 PCM DIGITAL RECORDER

The Model 4100 recorder is used to log PCM data. The data is decommutated into words and is stored along with additional timing and data keying.

8mm DIGITAL TAPE STORAGE

The Model 4100 recorder uses inexpensive 8mm data quality tapes for high volume digital storage. The tape containing the data is written by the EXABYTE 8505 tape drive in an ANSI standard file and is compatible with the tape backup devices sold by many vendors for disk backup purposes.

NO OTHER GROUND SUPPORT EQUIPMENT NEEDED

The data on the tape can be read by high level languages. The data is decommutated when recorded, and is presented to the software program in a word stream from the digital tape media.

IRIG-B TIME STAMPING OF THE DATA

The Model 4100 records the IRIG-B bit stream which can be used to correlate the data on the tape with external events. It also has an internal 1 millisecond time generator in case the IRIG-B data stream is not present.

ANALYSIS OF THE DATA

The tape containing the data can be read by computers using industry standard 8mm tape drive units which use the EXABYTE 8505 tape drives. For more information about commercial software analysis programs, contact Design Analysis Associates.

PACKAGING

The HVLOG Model 4100 comes in a desktop version with an IBM Compatible 486 PC. It also comes in the HVLOG Ruggedized enclosure.

INPUTS

Two PCM NRZ-L Data streams with Clock
  Word Size: 8 to 32 Bits
  Frame Size: 1 to 65,535 Words
One Serial IRIG-B data Stream
8 Digital Discretes

SOFTWARE SUPPORT

The unit comes with the software to configure the unit for the different PCM formats, record, view, and transfer the data to other media. Commercial data analysis software is available.

Specifications for Rugged HV4100-RGB

Dimensions:
  Enclosure:
    Height: 6.75 Inches
    Width: 10.19 Inches
    Depth: 15.38 Inches
  Enclosure and Mount:
    Height: 8.59 Inches
    Width: 11.69 Inches
    Depth: 19.88 Inches
Weight (Including Mount)
  Net: 26 lb.
  Shipping: 38 lb.
Power:
  16 to 36 Volts DC
  70 Watts
  UPS power supply will power the unit for 10 minutes if primary power is lost.
Environment:
  Temperature:
    Operating: 5 deg. C to 40 deg. C
    Storage: -20 deg. C to 85 deg. C
  Humidity:
    20% to 90% Non Condensing
Vibration:
  The unit is shipped with shock mounts that limit the shock and vibration to the unit for use on helicopters. Other shock mounts are available on request.
Mounting Base:
  Barry Controls ALOA SERIES
  (Military Standard Number: MS91405-B1C)
Logging Media:
  Data Quality 8mm High Quality Video Tape Cartridge
  (3.7 X 2.5 X 0.6 Inches)
Data Rate:
  The HV4100 logs each PCM word into a 32 bit word on the Tape for word sizes 24 bits and lower. For word sizes 25 to 32 bits, two 32 bit words are stored on the tape. The HV4100 can log a 32 bit word every 10 Microseconds.
Reliability:
  Non-recoverable error rate:
    Less than one in 10**13 bits read.
Preventative Maintenance:
  Periodic Tape Head Cleaning

Specifications for Desktop HV4100-DT

Dimensions:
  Enclosure:
    Height: 7 Inches
    Width: 25 Inches
    Depth: 25 Inches
  Weight:
    Net: 35 lb.
    Shipping: 40 lb.
Power:
  120 Volts AC +/- 10%
  70 Watts
  (220 VAC Supply available on request)
Environment:
  Temperature:
    Operating: 41 deg. F to 104 deg. F
    Storage: -4 deg. F to 185 deg. F
Mounting:
  Desk Top
  Tower
Interface:
  Two PCM NRZ-L data streams with clock. (Single ended or RS-422)
  IRIG-B serial data stream.
  26 Digital Discretes
Logging Media:
  Data Quality 8mm High Quality Video Tape Cartridge
  (3.7 X 2.5 X 0.6 Inches)
Data Rate:
  The HV4100 logs each PCM word into a 32 bit word on the Tape for word sizes 24 bits and lower. For word sizes 25 to 32 bits, two 32 bit words are stored on the tape. The HV4100 can log a 32 bit word every 10 Microseconds.
Reliability:
  Non-recoverable error rate:
    Less than one in 10**13 bits read.
Preventative Maintenance:
  Periodic Tape Head Cleaning

MODEL 5100 485 UTILITY BUS DIGITAL RECORDER

The Model 5100 recorder is used to log the 485 UTILITY bus data. The data is formatted into words and is stored along with additional timing and data keying.

8mm DIGITAL TAPE STORAGE

The Model 5100 recorder uses inexpensive 8mm data quality tapes for high volume digital storage. The tape containing the data is written by the EXABYTE 8505 tape drive in an ANSI standard file and is compatible with the tape backup devices sold by many vendors for disk backup purposes.

NO OTHER GROUND SUPPORT EQUIPMENT NEEDED

The data on the tape can be read by high level languages. The data is formatted when recorded, and is presented to the software program in a word stream from the digital tape media.

IRIG-B TIME STAMPING OF THE DATA

The Model 5100 records the IRIG-B bit stream which can be used to correlate the data on the tape with external events. It also has an internal 1 millisecond time generator in case the IRIG-B data stream is not present.

ANALYSIS OF THE DATA

The tape containing the data can be read by computers using industry standard 8mm tape drive units which use the EXABYTE 8505 tape drives. For more information about commercial software analysis programs, contact Design Analysis Associates.

PACKAGING

The HVLOG Model 5100 comes in a desktop version with an IBM Compatible 486 PC. It also comes in the HVLOG Ruggedized enclosure.

INPUTS

One UTILITY BUS 485 Digital Bus
One Serial IRIG-B data Stream
10 Digital Discretes

SOFTWARE SUPPORT

The unit comes with the software to record, view, and transfer the data to other media. Commercial data analysis software is available.

Specifications for Rugged HV5100-RGB

Dimensions:
  Enclosure:
    Height: 6.75 Inches
    Width: 10.19 Inches
    Depth: 15.38 Inches
  Enclosure and Mount:
    Height: 8.59 Inches
    Width: 11.69 Inches
    Depth: 19.88 Inches
Weight (Including Mount)
  Net: 26 lb.
  Shipping: 38 lb.
Power:
  16 to 36 Volts DC
  70 Watts
  UPS power supply will power the unit for 10 minutes if primary power is lost.
Environment:
  Temperature:
    Operating: 5 deg. C to 40 deg. C
    Storage: -20 deg. C to 85 deg. C
  Humidity:
    20% to 90% Non Condensing
Vibration:
  The unit is shipped with shock mounts that limit the shock and vibration to the unit for use on helicopters. Other shock mounts are available on request.
Mounting Base:
  Barry Controls ALOA SERIES
  (Military Standard Number: MS91405-B1C)
Logging Media:
  Data Quality 8mm High Quality Video Tape Cartridge
  (3.7 X 2.5 X 0.6 Inches)
Interface:
  One UTILITY BUS 485 Digital Data Bus.
  IRIG-B serial data stream.
  8 Digital Discretes
Data Rate:
  The HV5100 logs each UTILITY BUS word into three 16 bit words on the Tape. The recorder is capable of supporting the full bandwidth of the Utility BUS.
Reliability:
  Non-recoverable error rate:
    Less than one in 10**13 bits read.
Preventative Maintenance:
  Periodic Tape Head Cleaning

Specifications for Desktop HV5100-DT

- Dimensions:
  - Enclosure:
    - Height: 7 Inches
    - Width: 25 Inches
    - Depth: 25 Inches
  - Weight:
    - Net: 35 lb.
    - Shipping: 40 lb.
- Power:
  - 120 Volts AC +/- 10%
  - 70 Watts
  - (220 VAC Supply available on request)
- Environment:
  - Temperature:
    - Operating: 41 deg. F to 104 deg. F
    - Storage: -4 deg. F to 185 deg. F
- Mounting:
  - Desk Top
  - Tower
- Interface:
  - One UTILITY BUS 485 Digital Data Bus.
  - IRIG-B serial data stream.
  - 10 Digital Discretes
- Logging Media:
  - Data Quality 8mm High Quality Video Tape Cartridge
  - (3.7 X 2.5 X 0.6 Inches)
- Data Rate:
  - The HV5100 logs each UTILITY BUS word into three 16 bit words on the Tape. The recorder is capable of supporting the full bandwidth of the Utility BUS.
- Reliability:
  - Non-recoverable error rate:
    - Less than one in 10**13 bits read.
- Preventative Maintenance:
  - Periodic Tape Head Cleaning

Signal Measurement System-Abstract

Introduction

To determine the effectiveness of existing and emerging EW systems in today's sophisticated threat environment, the systems must be tested in scenarios that realistically simulate the complexity and density of the electronic order of battle. Evaluating the results of such tests is impossible without an accurate record of the test scenario and the system's response.

Amherst Systems' Signal Measurement System (SMS) is a real-time, wide-band, electromagnetic environment verification system which has been designed to meet the needs of EW Systems Test and Evaluation. It can be used in anechoic chambers, on ranges or in bench test configurations.

The unique wide-band features of the SMS enables it to reliably characterize all the signals in the electromagnetic environment. In addition to the programmed signals, inadvertent and erroneous emissions from test equipment or the System Under Test (SUT) are detected, characterized and reported in real-time to the test controller.

The SMS identifies the measured signals and correlates ECM transmissions from the system under test with programmed threat emissions. 'Quick-look' displays keep the test operator informed of the test progress. Test events are stored to allow off-line analysis and an accurate evaluation of the performance of the SUT.

Description

The SMS is comprised of four sub-systems: input, receiver, signal processor and control computer. The sub-systems are comprised of commercially available test instrumentation and hardware supported by application specific signal analysis software.

The antenna configuration is application specific; in a range application the input to the SMS is derived directly from the antenna feeds of the threat simulators; in a chamber application from the receiving antennas located at each threat site and in a bench-test situation inputs are directly coupled from the RF input and output of the SUT.

The wide-band receiver has an instantaneous bandwidth covering the SUT's entire operating frequency. All signals in the environment will be characterized, time-tagged and handed off to the signal processor.

The signal processor deinterleaves the digitized pulse trains and identifies each emitter. The signal sorting and identification software is designed to perform under conditions of severe pulse train corruption. In the event that an ECM emission is detected further detailed analysis of timing, phase, frequency and amplitude is performed. Each ECM emission and mode change is time-tagged and is correlated with a corresponding threat event.

The control computer provides on-line monitoring of the progress of the test and the SUT's performance via programmable 'Quick-look' displays. This provides the capability for the real-

time interception and analysis of test and SUT performance anomalies. If such anomalies occur then intervention in the test process can prevent costly test reruns.

Conclusions

To accurately predict the performance of EW systems in today's sophisticated electronic battlefield, dynamic testing in realistic scenarios is required. The verification of test conditions and SUT performance can be achieved with the innovative control and application of existing technology.

Recording and post test analysis of the test environment and SUT performance can pay dividends by reducing test time and resolving disputes between equipment manufacturers and test authorities.

Section 1
INTRODUCTION

During the development and service life of an EW system it is subjected to various developmental and operational tests designed to evaluate its capability to perform in diverse tactical situations. An illustration of the life cycle for a DoD system is shown in Figure 1-1 together with a mapping of the type of testing to which it is subjected to ensure that it meets its operational requirements.

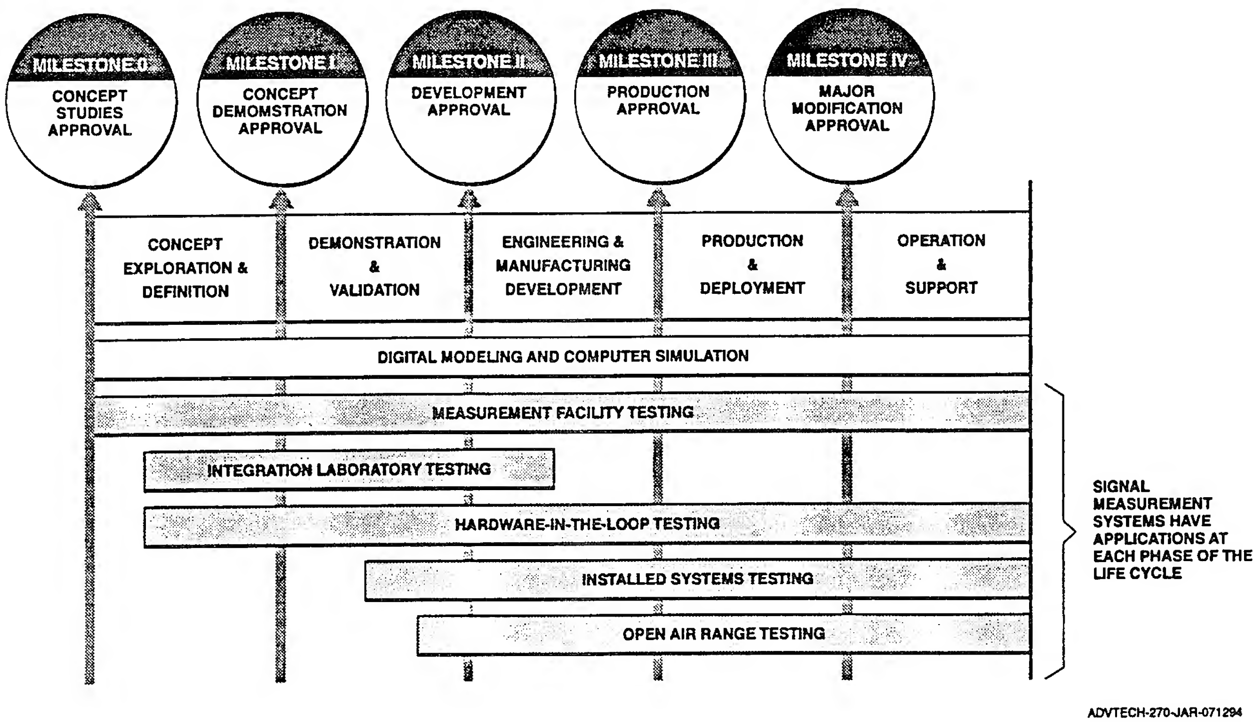


Figure 1-1 EW System Life Cycle and Test Requirement.

Some of the most rigorous testing that a system is subjected to is conducted using electromagnetic environment simulators. These tests involve the generation of a synthetic electromagnetic environment that simulates, in all characteristics, the various radar threat environments that the system might encounter during a mission. Systems are routinely tested against scenarios that contain mixtures of the most modern and the most mature threats from the Red, Blue and Grey inventories. As can be seen from Figure 1-1 the systems are subjected to these tests at all stages of their life cycle, from concept to service operation.

In order to ensure the validity of the tests the characteristics of the environment and the response of the system under test must be measured and recorded. Typically these measurements are performed using narrow band receivers and Spectrum Analyzers that tune to the frequency of a unique signal and measure its parameters. This technique has a number of inadequacies that can adversely impact the analysis of the test results and lead to an inaccurate assessment of the performance of the system. Some of these problem areas are as follows; a). Emmissions from third parties that are within the test frequency range and can influence the performance of the system under test are not detected and recorded (of particular concern on open air ranges). b). Inadvertent and erroneous emmissions from the test equipment and the system under test are not detected and can also influence the test results. c). Timing relationships between a threat and the ECM technique applied by the system under test are not correctly quantified. These relationships can be fundamental to the effectiveness of the system but generally are not measured under realistic engagement conditions.

This paper presents an alternative measurement system that addresses the inadequacies of the measurement systems currently used in test establishments. The proposed system also provides the capability to perform real time monitoring and evaluation of the test progress and to record the test data for post test analysis and test reconstruction. The system utilizes COTS components operating under control of proprietary software, some of which is in use in existing test systems.

Section 2
DESCRIPTION

The Amherst Systems' Signal Measurement System is comprised of the following sub-systems: input, receiver, signal processor and control computer. The sub-systems are comprised of commercially available test instrumentation and hardware supported by application specific signal analysis software. A system block diagram is shown in Figure 2-1 below.

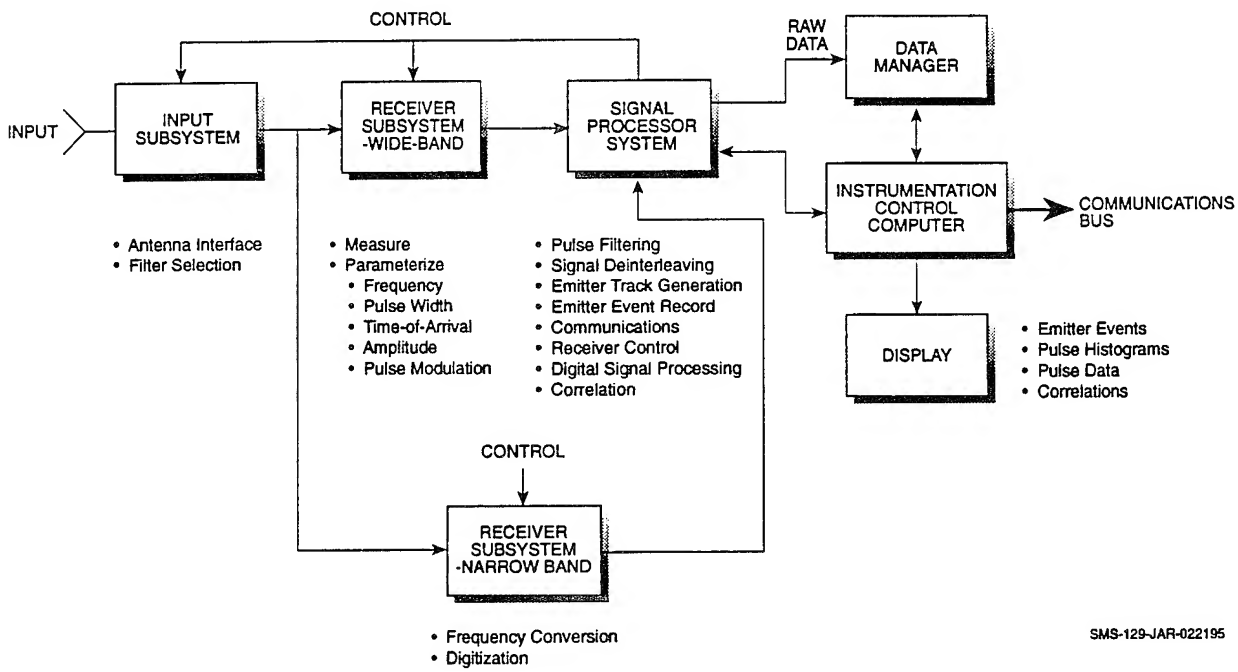


Figure 2-1 System Block Diagram

The **input subsystem** configuration is application specific. In a bench-test situation, inputs are directly coupled from the RF input and output of the SUT. Samples are taken from each input port of the SUT and directed to the receiver subsystem for parameter measurement and angle-of-arrival (AOA) determination. In a range configuration, the environment is sampled via a broad band antenna, and in an anechoic chamber configuration the inputs are provided by a set of antennas distributed around the test chamber. Switched amplifier/filter assemblies are provided to assist in the control of the signals that stimulate the receiver. These assemblies are controlled by the signal processor to remove CW and high duty cycle signals to prevent receiver saturation and to improve the probability of intercept of low duty cycle emitters.

The **receiver subsystem** is comprised of two types of receivers, one a tunable narrow band receiver and the other a wide band receiver. The wide band receiver utilizes commercially available Instantaneous Frequency Measurement (IFM) receivers having wide instantaneous bandwidths that together cover the entire operating frequency range of the SUT. The

specification of a typical IFM is shown in Table 2-1. These receivers have high sensitivity, wide dynamic range and high probability of detection of any signal in their passband. Each input pulse is characterized by digitization of its frequency, amplitude and pulse width parameters and is time-tagged by the receiver. These pulse descriptor words are then handed off to the signal processor. In test situations that require angle-of-arrival measurement (e.g. bench testing of an EW system), special purpose hardware is provided to determine the AOA of each signal. Fast A/D converters in each input feed digitize the amplitude of the pulses to an accuracy of 0.5 dB and comparators determine the AOA.

Table 2-1 Typical IFM Specification

	Min	Nominal	Max	Units
Operating Frequency Frequency range over which this specification applied	6		18	GHz
Frequency Resolution (13 bits) Value of Least Significant Bit (LSB)		1.5		MHz
Frequency Accuracy (RMS) RMS frequency error of measurements across operating frequency range		3		MHz
Binary Zero Frequency Frequency at which the digital outputs are all Os (TTL- Low)		5866		MHz
Unambiguous BW		12288		MHz
RF Input Dynamic Range RF input range over which this specification applies	-60		+5	dBm
Amplitude Resolution (8 Bits) Value of LSB		.3125		dB
RF Pulse Width Range RF input pulse width range over which this specification applies, CW signal is defined as pulse width greater than 102.4 μsec	0.1		102.4	μsec
RF Pulse Width Resolution (11 bits) Value of LSB		50		nsec
RF Pulse Width Measurement Error (1) One resolution cell or 10% of pulsewidth, whichever is larger	±50		10%	nsec
Time of Arrival Resolution (TOA of 16 Bits) Value of LSB		0.5		usec
Signal-to-Noise Ratio The RF input ratio of the signal power to noise power, constrained by the operating frequency range, for which Probability of Detection and False Alarm are valid	3			dB

Probability of Detection Probability that a RF pulse within the operating frequency range is detected and reported within the specification limits	99			Percent
Probability of False Alarm Probability that the receiver issues a detected signal with parameters not within specification			5	Per Second
Throughput Time Time from trailing edge of RF to leading edge of next RF			250	nsec
RF Input VSWR			2.2:1	

The narrow band receiver is comprised of a tunable superhet receiver that outputs an IF signal which is input to a high speed analog-to-digital converter (ADC). The digitized data is input to a Digital Signal Processing (DSP) board where software routines analyze the frequency, phase, amplitude and time parameters of the signal. Complex intra-pulse and inter-pulse modulated waveforms that are characteristic of ECM waveforms are characterized using this receiver.

The narrow band receiver is under the control of the signal processor subsystem. Tuning, bandwidth and processing instructions are passed from this processor to the receiver depending upon the conditions of the RF environment and the test strategy. When operating in an integrated test facility where the RF environment is being scripted by a computer simulation, *a priori* information from the simulation script and the threat simulator is available to assist in the control of the receiver.

The **signal processor subsystem** is based on the design of the Amherst Receiver Processor System (ARPS) and is comprised of commercially available processor boards that execute signal processing software designed by Amherst Systems. This software was designed specifically to deinterleave pulse trains comprised of highly corrupted data (i.e., pulse trains with missing pulses, multipath effects and incorrectly parameterized pulses). The signal processor system employs a multi-processor architecture to achieve high pulse throughput rates (in excess of one million pulses per second) and a simplified means of maintenance and evolutionary improvement. Each processor performs a dedicated function or a few closely related functions. The architecture allows tailoring of the functions assigned to the individual processors to match the data throughput requirements. For exceptionally high throughput applications, multiple processors are assigned to perform each time-critical task. Key features of the processor system are:

- Concurrent asynchronous operation of the individual processors
- True pulse-by-pulse processing
- High tolerance to corrupted data.

The modular design of the processor subsystem supports expansion to an arbitrarily large number of processors as the requirements for the system expands and for replacement of one or all of the

existing processors with more powerful processor boards as required. A functional diagram of the signal processor is shown in Figure 2-2.

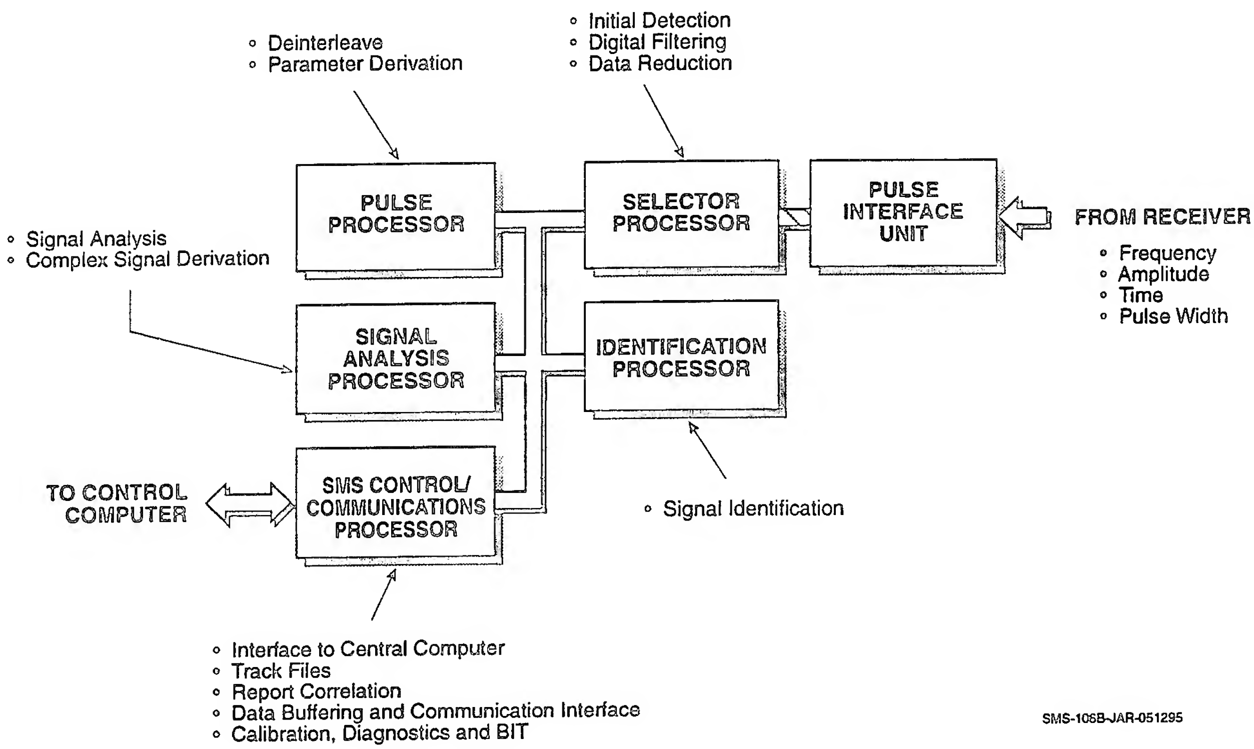
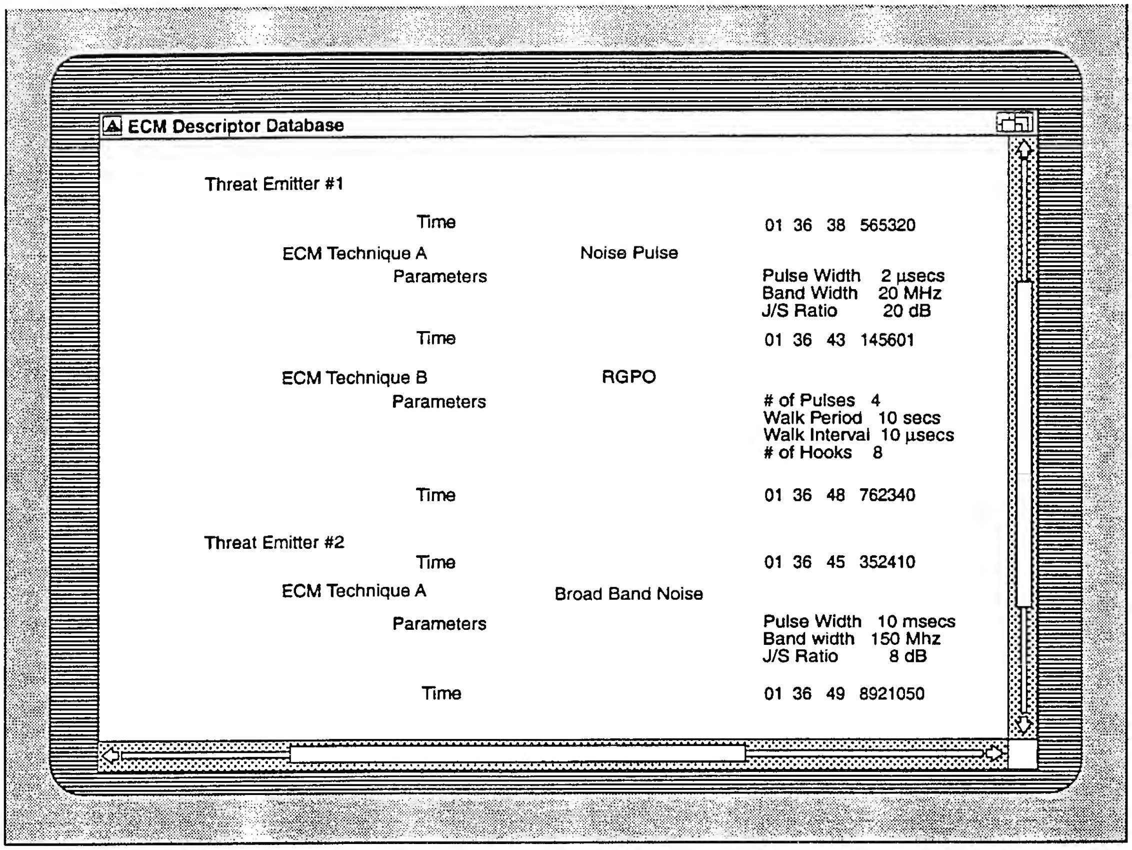


Figure 2-2 Signal Processor Functional Diagram

The digitized pulses from the wide-band receiver are sorted into discrete pulse trains and identified as specific emitters. In the event that an ECM emission is detected, further detailed measurements of timing, phase, frequency and amplitude are performed in the narrow band receiver. Each ECM emission and mode change is time-tagged and correlated with a corresponding threat event. A sample report correlating the ECM and the threat that it is countering is shown in Figure 2-3. If the processor can be interfaced to the SUT data bus, then realtime comparisons can be made between the SMS measured data and the SUT measured data.



ADVTECH-260-MJM-120394

Figure 2-3 ECM Correlation Report

The **control computer** enables the test operator to configure the hardware and software for specific tests. Databases can be entered into the system or modified via the keyboard as required. An important feature of the SMS is the on-line monitoring of the progress of the test and the performance of the SUT via the programmable "Quick-look" displays of the control computer. By interception and analysis of test equipment and SUT performance anomalies, costly test reruns can be avoided. Test data is recorded for post test analysis.

A functional diagram of the SMS is shown in Figure 2-4. This design covers the 50 MHz to 18 GHz frequency band, using three IFMs for measurement of the threat environment from 0.5 to 18 GHz and a narrow band superhet receiver for the 50 MHz to 500 MHz band. In addition to providing the low band frequency extension, this narrow band receiver provides the capability to monitor and characterize ECM techniques over the full frequency coverage of the system (50 MHz to 18 GHz). Interfaces can be provided for the SUT data bus, inputs from the script of the test scenario and for a realtime data interchange from the threat simulator. These inputs enable the test operator to a). ensure that the environment, as measured by the SMS, agrees with the programmed output expected from the simulator, and b). to monitor the performance of the SUT. This is not an essential feature of the system, but it does provide a potentially time saving check on the progress of the test.

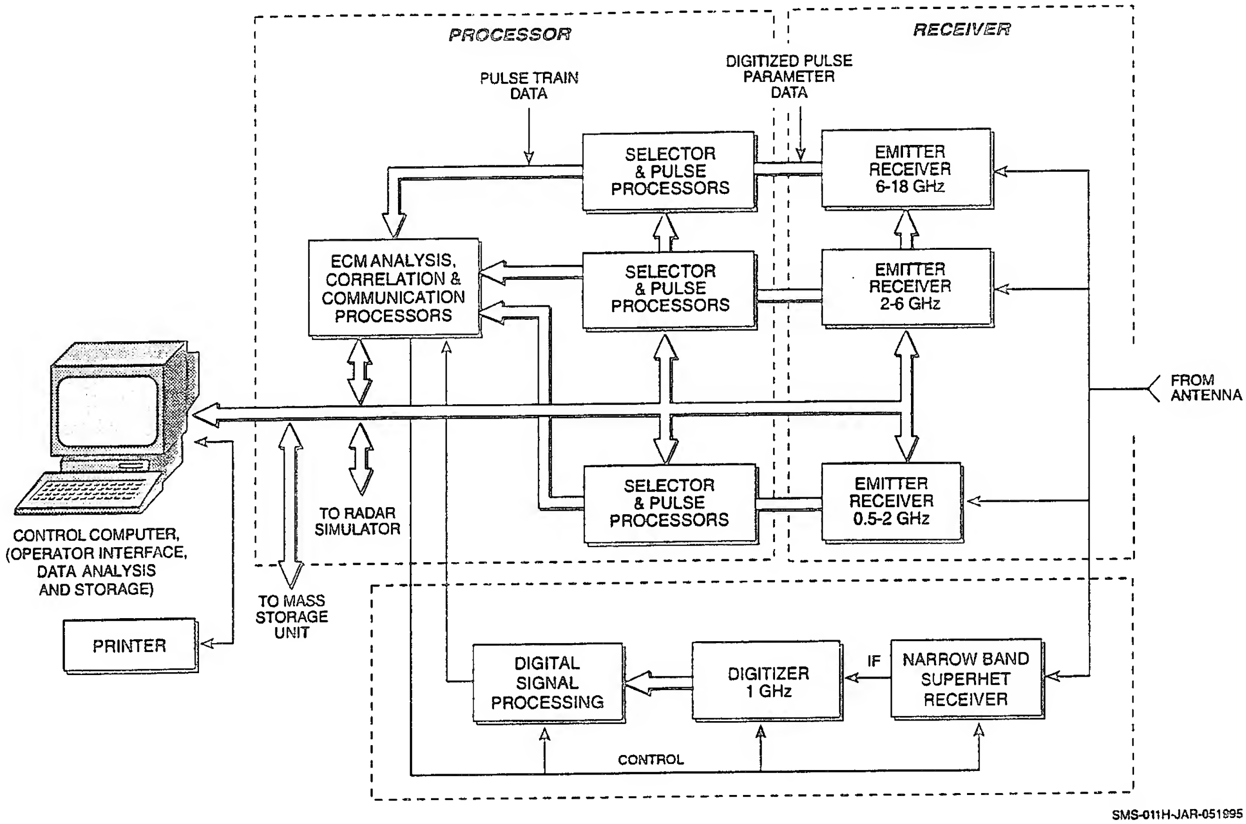


Figure 2-4 System Functional Diagram

Some of the functions of the SMS are:

- Verification of the parameters of each emitter taken one at a time.
- Verification of the scenario generation.
- Determination of scenario dynamics such as:
  - pulse densities
  - number of active emitters
  - frequency dispersion of the scenario
  - timing of emitter events such as ON/OFF and mode changes
- Verification of the threat identification process of the SUT.
- Identification of ECM threat technique.
- Validation of the ECM technique for the perceived threat.
- Correlation of the ECM technique with the threat.

The Signal Measurement System is a fully automated system. Its wide dynamic range and wide instantaneous frequency bandwidth ensure that all active signals will be detected and characterized.

Section 3
OPERATION

An operational representation of the SMS in an anechoic chamber configuration is shown in Figure 3-1. Interfaces are shown that enable the SMS to compare the measured environment with the threat simulator program and the SUT data bus information. Samples of the threat environment are coupled off the RF feeds to the transmit antennas or are received via antennas strategically positioned within the chamber. The ECM transmissions from the System Under Test are detected as radiated signals via antennas installed in the chamber. Alternatively, the ECM waveforms may be detected by direct coupling off the transmit system or by using antenna hats on the ECM system antennas.

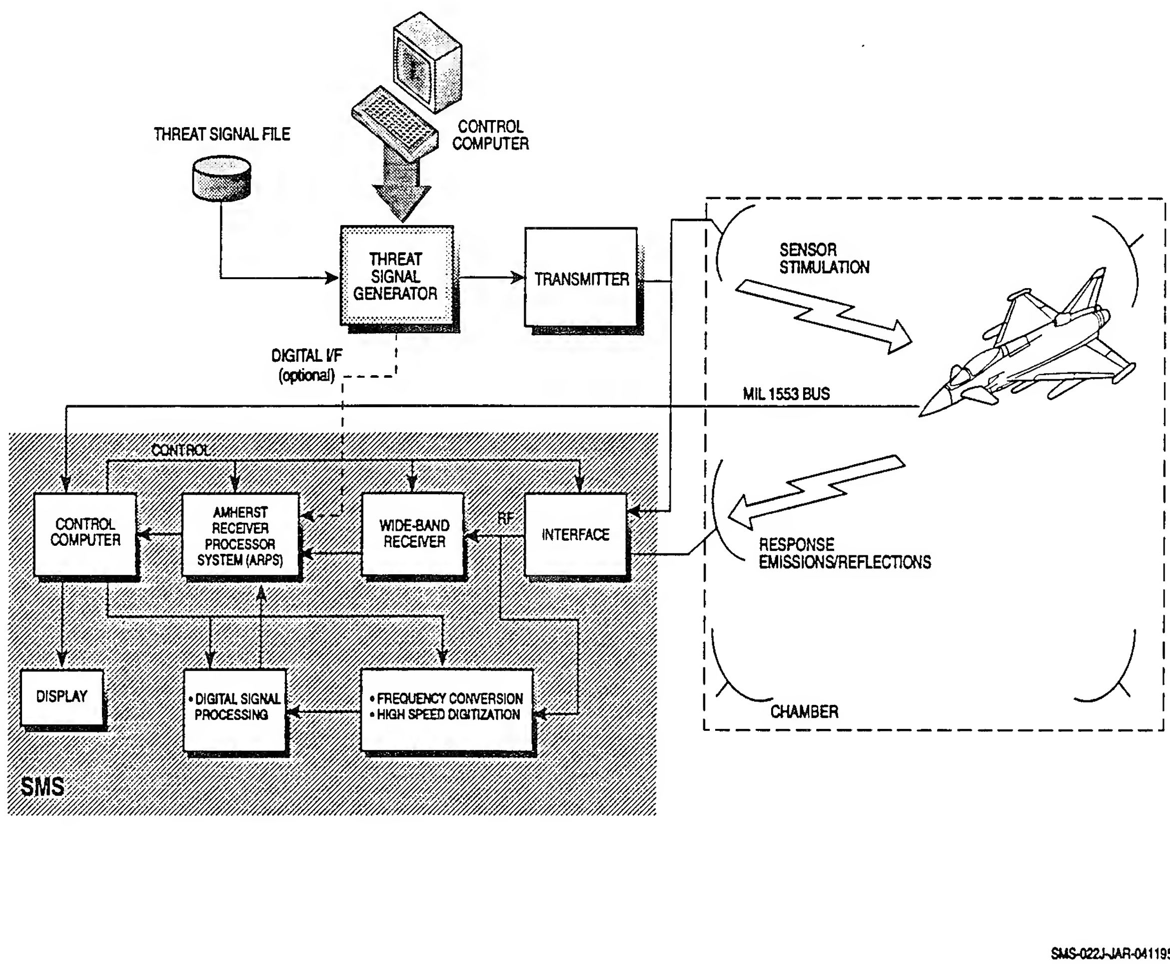


Figure 3-1 Anechoic Chamber Operation

EASILY PROGRAMMABLE WEAPON SIMULATOR

Code 41L300D
Naval Air Warfare Center
China Lake, CA 93555-6001
Telephone 619-939-4946, FAX 619-939-9944

(41L300-SIM-TFWG-4712-00-U)

1.0 Introduction

The AV-8B Weapons System Support Facility (WSSF) required a new system to simulate the AV-8B Weapons because of upgrade and maintenance difficulties with older equipment. This new Weapons Simulator(WPNSIM) was essential for lab testing of both the Operational Flight Programs (OFPs) and the Stores Management System (SMS) software and hardware components. The simulator had to be easy to program and debug, and have the ability to support rapid prototyping and testing efforts. Development of this system was accomplished in 11 man-months with material costs of only $40,000.

As an example of the WPNSIM's ability to meet the rapid prototyping and debug requirement, a request was made to modify the simulator on the SMS bench to simulate the "AV-8B hooks bouncing closed" phenomenon. This change was investigated and implemented on the AV-8B Weapons Simulator in one week, with no parts' costs. (Change estimates for other simulators were for several months.)

2.0 System Description

The new WPNSIM uses a commercial Versa Module Eurocard (VME) Bus and a RadiSys Corporation 386 PC as the Graphical User Interface (GUI) host and system controller. The GUI is written in Borland C++ with a Zinc Interface windows library, and communicates with the custom station controller Central Processing Units (CPUs). The station controller CPU user program is written in Microtec C and stored in Electrically Erasable Programmable Read Only Memory (EEPROM).

Each station controller connects to a customized Input/Output (I/O) board that simulates weapon signals such as bomb rack hooks, missile identification discretes, and seeker head signals. Each station controller CPU has a built-in Debug/Monitor program and a RS-232 port (Figure 1). a terminal or emulator plugged into this port lets the station controller becomes a stand-alone computer. The Debug/Monitor program can read and write to memory, find checksums, and load the user program into EEPROM, greatly simplifying prototyping and debugging procedures.

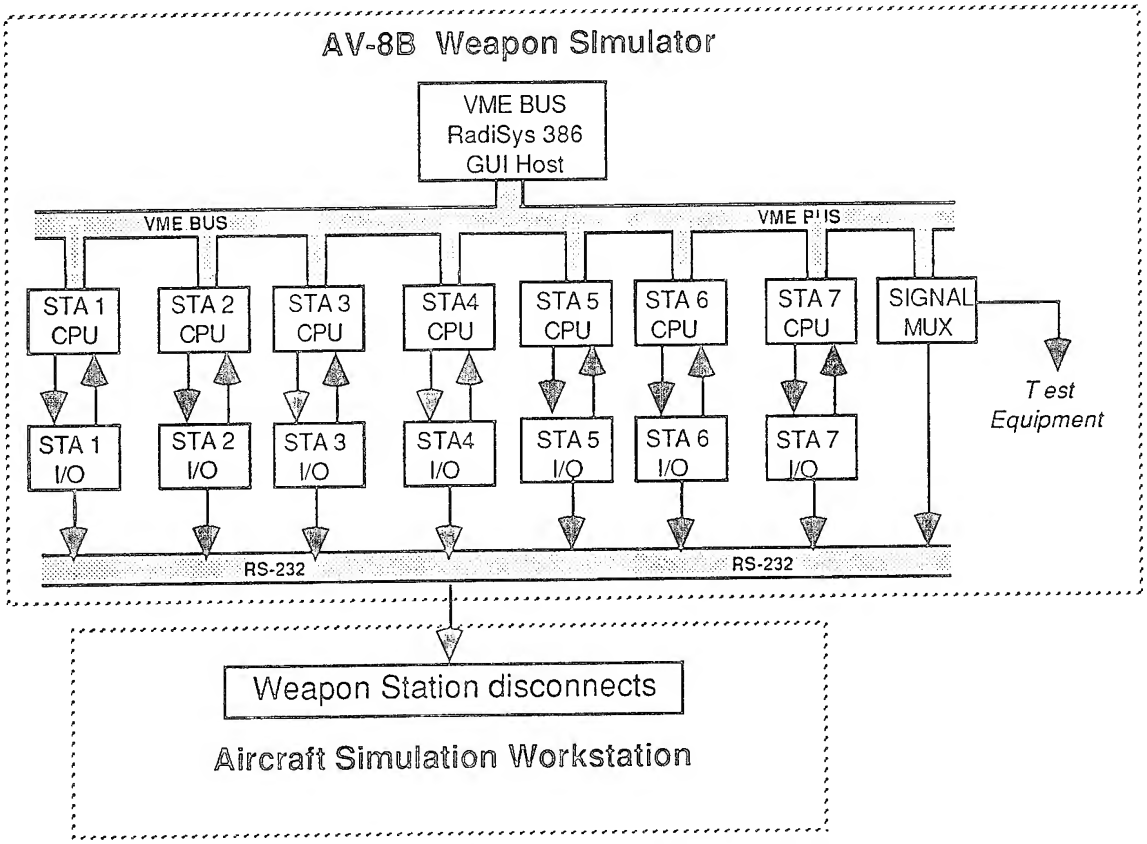


Figure 1. Weapon Simulator CPU and I/O Layout.

The WPNSIM is used with each of the AV-8B WSSF workstations to simulate external stores for the AV-8B simulation. The WPNSIM is connected to the workstation by connecting the WPNSIM's Station Connect Panel cables to the workstation's Weapons Disconnect Panel (Figure 2). The Weapons Disconnect Panel duplicates the aircraft's pylon connectors. The following paragraphs describe the hardware and software components that make up the WPNSIM.

3.0 WPNSIM Hardware Components

The WPNSIM is constructed on a roll-around cart, enabling transportability from one AV-8B workstation to another. A standard 14-inch VGA color monitor and Trak101 keyboard are the user interface to the WPNSIM software. The RadiSys ECP®-3P 386 WPNSIM system controller is a DOS-based processor, with a 40 megabyte hard drive and an attached 3-1/2-inch floppy drive to load new/updated programs. This processor stores the WPNSIM program.

The WPNSIM has eight custom station controller boards and eight custom station interface boards (7 per weapon station and 1 per gun) which allows the WPNSIM to communicate to or from the Stores Management Computer (SMC). A one-millisecond timer board supplies the timing for the Input Discrete Event Queue.

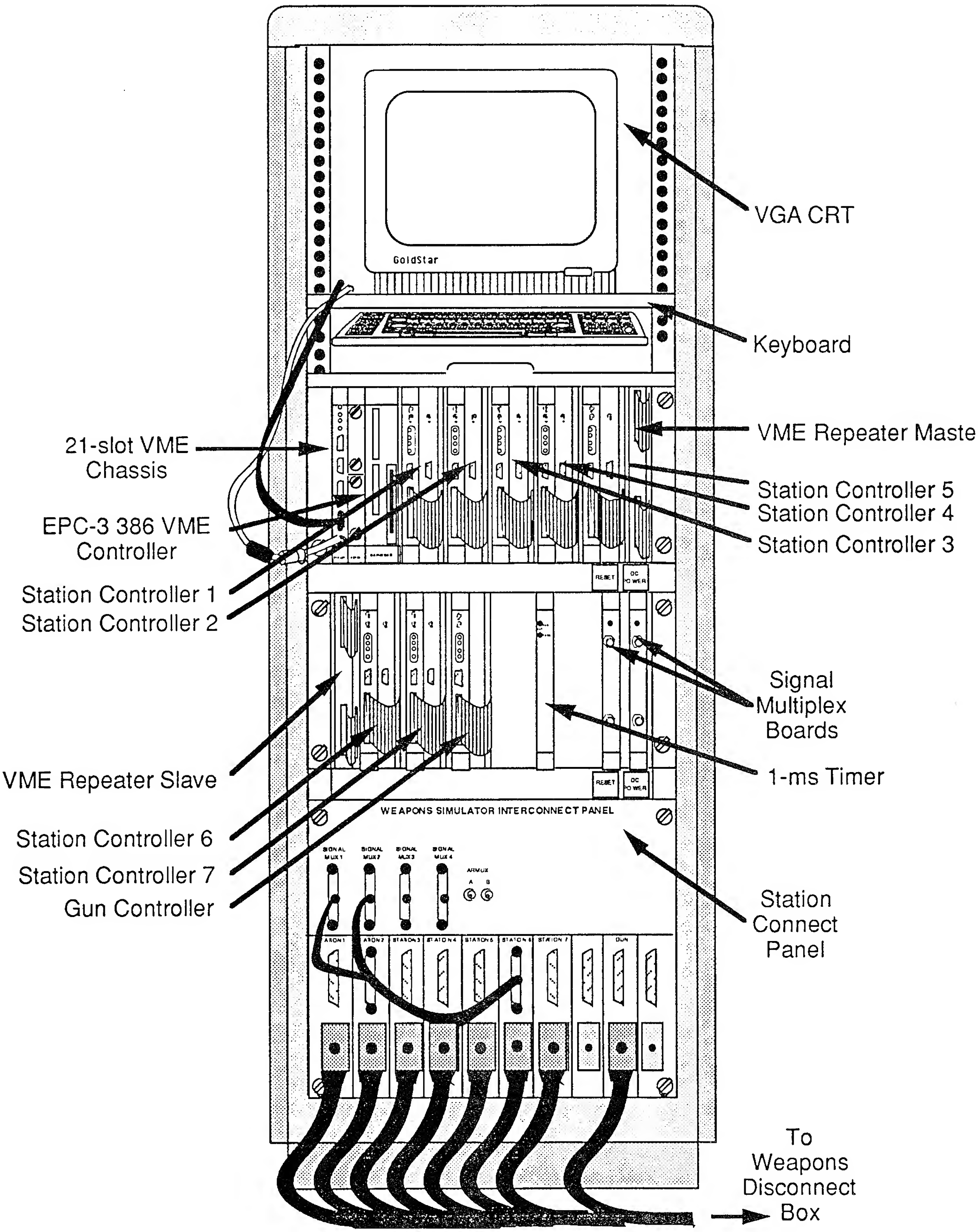


Figure 2. Weapon Simulator.

Up to four signal multiplexer boards are used by the WPNSIM. These signal multiplexer boards enables the Connect Bayonet Connector (BNC) jacks on the front of each board to capture any signal going to the WPNSIM. The VME Repeater Master/Slave boards connect the two VME chassis so the 386 processor can communicate with the boards in the bottom VME chassis.

4.0 WPNSIM Software Components

The WPNSIM software is written in 'C++' and runs on a RadiSys ECP®-3P 386 system. The software provides the signals needed to simulate the stores carried by the AV-8B at each weapon station. In addition, the WPNSIM provides the capability of monitoring and controlling signals between the simulated weapons and the Station Control Units (SCUs). These signals can be monitored internally as digital values or externally as analog signals through the built-in signal multiplexer. This multiplexer is also controlled by the WPNSIM software.

The user interface to the WPNSIM software is provided by Zinc's Interface Library. The Interface Library is used to design user-interactive windows.

5.0 WPNSIM User Interface

When the WPNSIM is powered up, the WPNSIM window (Figure 3) is displayed on the VGA monitor. This window is the main interface for the WPNSIM program.

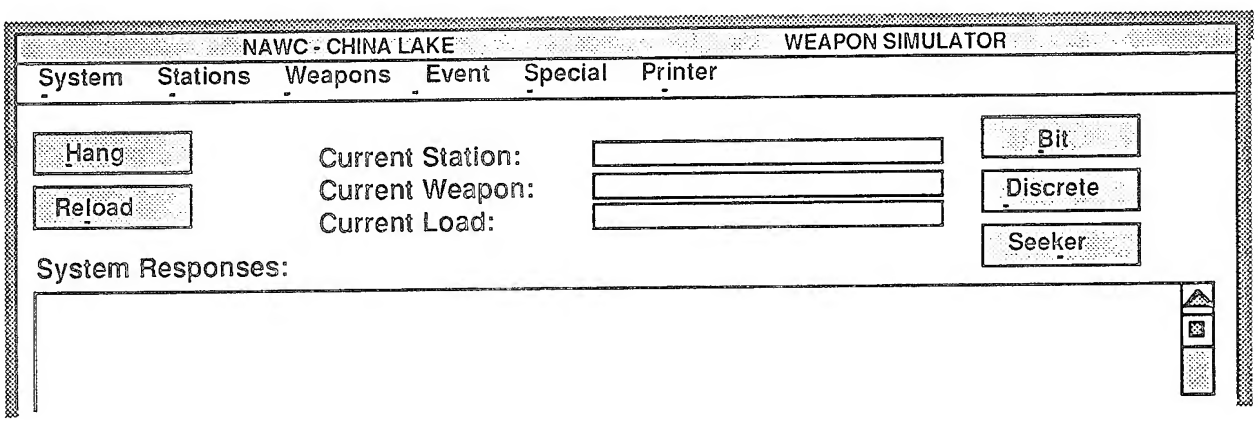


Figure 3. WPNSIM User Interface.

The Hang button, located at the upper left of the screen, creates a hung condition on the current station. When it is pressed the following message is displayed (*nnnn* values depend on the weapon):

Condition Codes: STAT[*nnnn*] ENABLE[*nnnn*] HANG[*nnnn*]
Full hang conditions ...

The Reload button, just below the Hang button, reloads a weapon on the current station after a weapons drop. When pressed the following message is displayed:

Reload complete

The BIT button, located at the upper right of the screen, performs a bit on the current station. When pressed the following message is displayed:

testing...please wait

When the test is complete, the system displays:

Success: BIT initiated...
***** BIT COMPLETE *****
No Errors
SimStatus: OK
MdlStatus: OK
ComStatus: READY
Version: 3
Checksum: 0x0000c901

If BIT is run when a weapon is selected the system displays the message:

***** BIT ABORTED *****
Cannot do BIT while running weapon model.

The Discretes button, just below the BIT button, provides access to the current station's input and output discrete signals. When selected the Discrete I/O window (Figure 4) overlays the WPNSIM window.

The Discrete I/O window consists of two pull-down menus and four list boxes. The Options pull-down menu enables you to refresh the screen after changing stations, While the Format pull-down menu enables you to choose between the weapons stations or the fuselage gun station. Options under Format determine the names to be given to the input discretes based on which type of station is selected.

The Station box is an input field that enables you to enter the weapon station number you wish to monitor. The Event Enables list displays the input signals and is used to determine which input signals will be monitored and recorded in the Discrete Event Queue accessed from the main menu. The Input Status list displays the current status of discrete input signals and is used to display the current status of discrete input signals. The Dynamic Control Enables list displays output signals and is used to enable or disable any of the output signals. A check mark next to a parameter name indicates that the parameter is enabled and may be altered by a weapon model. The Output Status list displays the status of discrete output signals and is used to set or clear a discrete signal. A check mark indicates a set signal.

The Seeker button, just below the Discretes button, allows control of the missile seeker head. The Sidewinder Extended Acquisition Mode (SEAM) can be enabled or disabled. Enabling SEAM allows the missile seeker head to nutate around the boresight to acquire a target. This button controls the modulation and peak audio voltage for the pilot's helmet, and slaves the boresight around the coordinates specified in phi and lambda. This button can only be accessed if you

have selected a Sidewinder or Sidearm on the current station. When you select this option, the Seeker Control window (Figure 5) overlays the WPNSIM window.

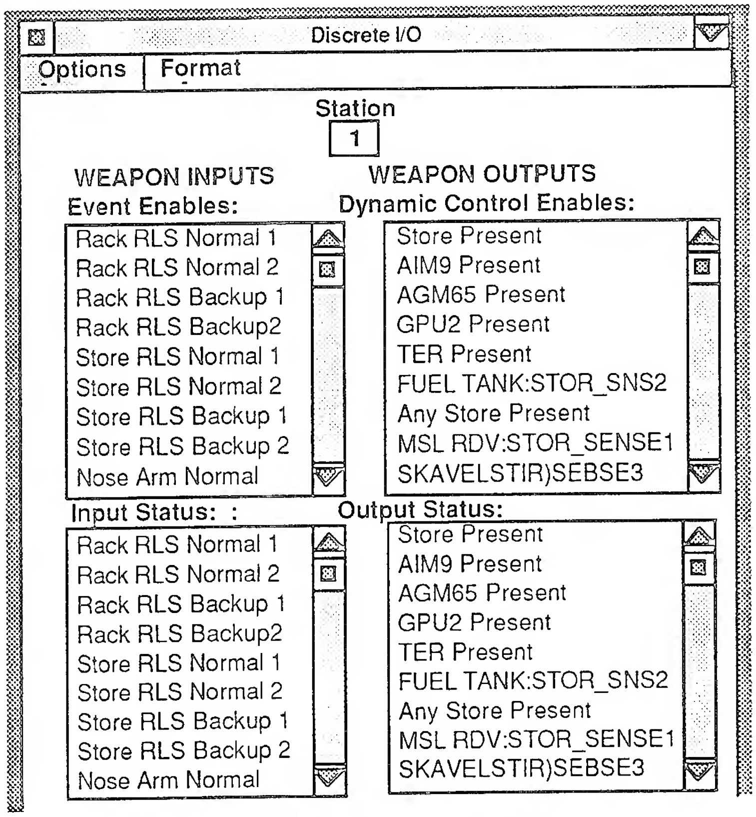


Figure 4. Discrete I/O Window.

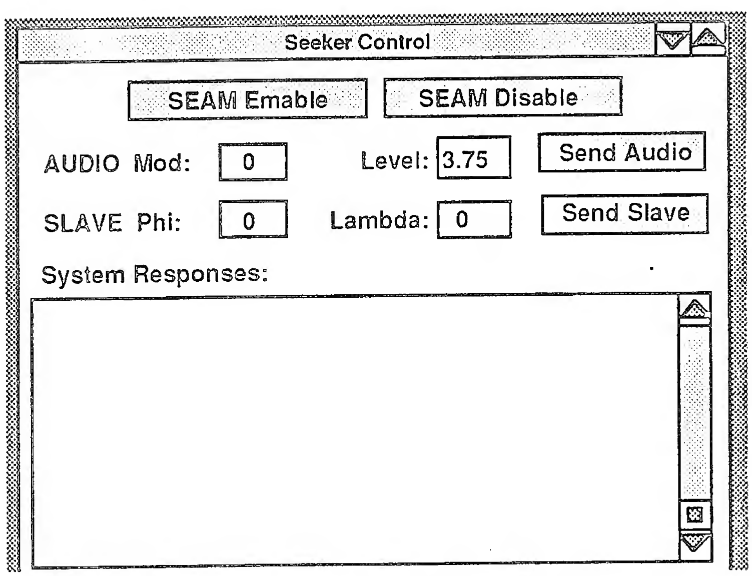


Figure 5. Seeker Control Window

The Seeker Control window components are the SEAM Enable button that implements SEAM; the SEAM Disable button that turns off the SEAM capability; the Audio fields Mod and Level control the audio tone in the pilot's helmet; the Phi

field specifies the angle around the boresight in degrees; the Lambda field specifies the angle from boresight in degrees; the Send Audio button transmits the audio level to the selected station's I/O board to control the level sent to the SMC. When the appropriate voltage is reached, the Lock-On relay in the bench will engage and stop the head nutation. The Send Slave button transmits the signal to slave the seeker head position.

5.1 System Pull-down Menu

The System pull-down menu provides access to commands that affect or report system wide conditions. The options described in Table I are displayed when the System pull-down menu is selected (see Figure 6) .

Table I. System Pull-down Menu Options.

Option	Description
About	Displays program, version, and developer information.
Welcome	Provides brief introduction to the WPNSIM program.
Inventory	Displays all stations, their weapon load, quantity, and status.
Unload All	Unloads all loaded weapon(s) on all stations.
Reload All	Reloads dropped weapon(s) on all stations.
Exit	Exits you from the WPNSIM program.
Bit	Performs BIT on all stations.
Debug	Performs debug on the WPNSIM software.

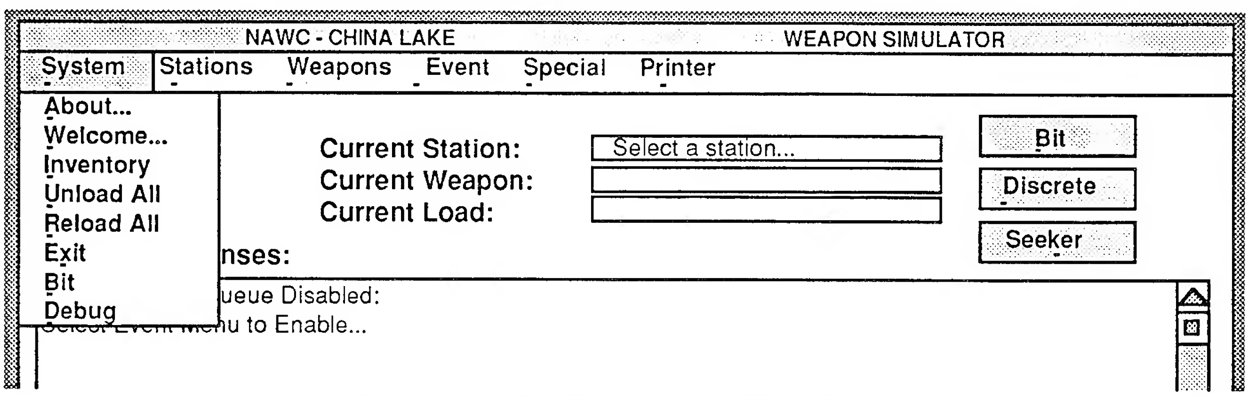


Figure 6. System Pull-Down Menu.

When selected the Inventory option is displayed. The displayed data includes the stations, loaded weapons, loaded weapons number or configuration, and the weapons state (e.g., hung). Figure 7 is an example of what might be displayed.

5.2 BIT Option

Selecting the BIT option performs BIT on all WPNSIM station boards. When you select this option, the following message is displayed:

TESTING...Please wait...

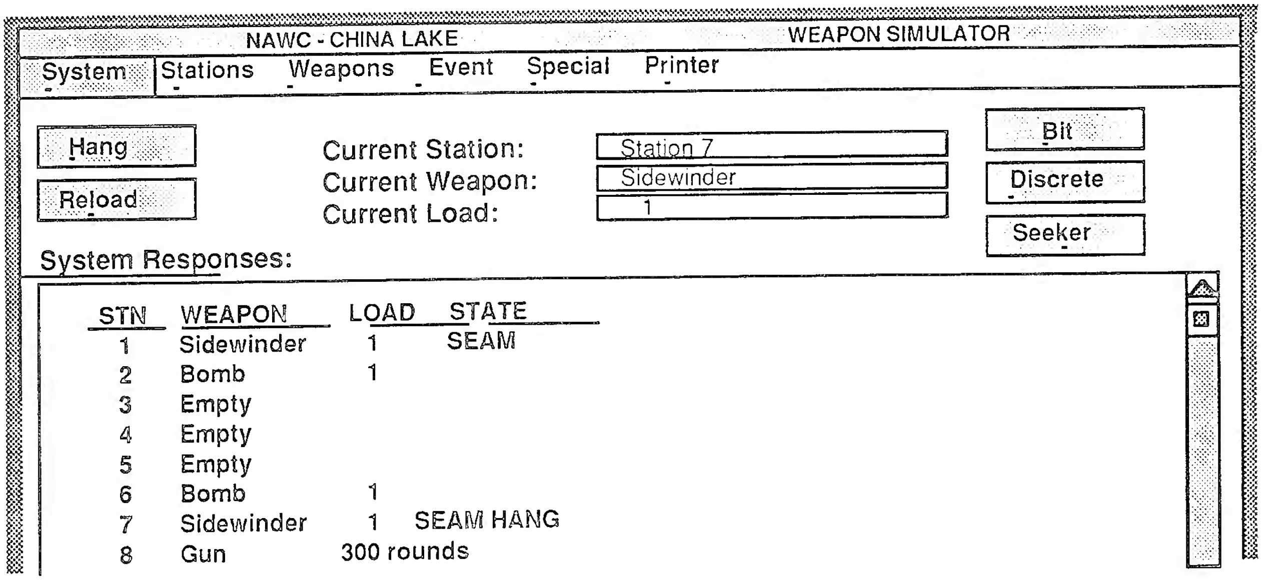


Figure 7. Sample Inventory Window.

If all stations passed, BIT, status messages are displayed (Figure 8).

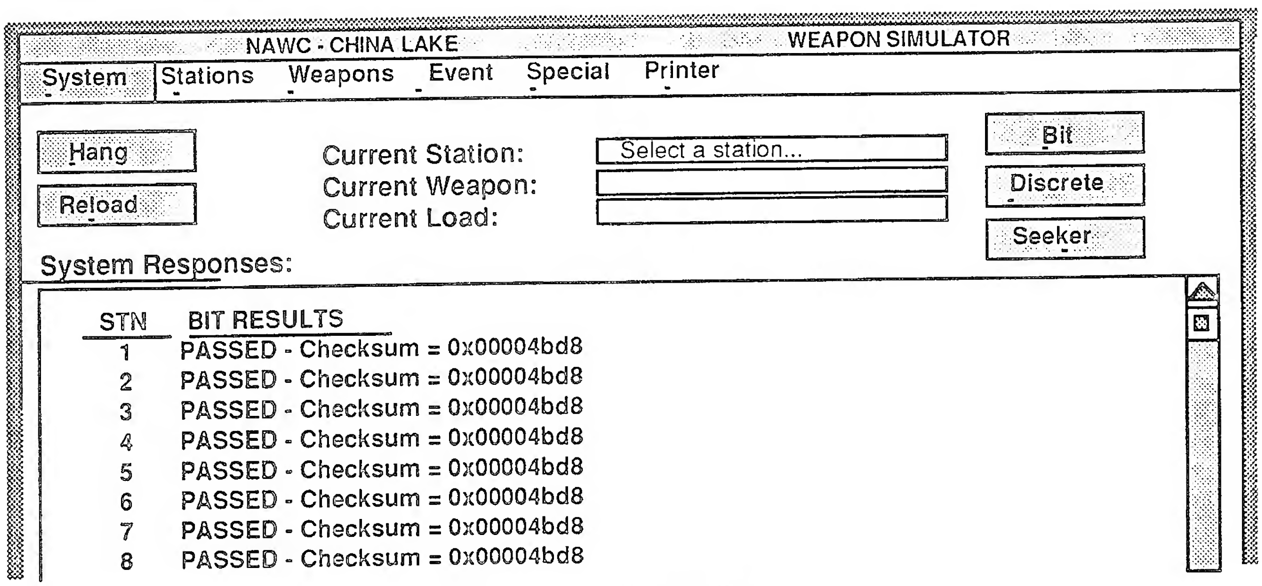


Figure 8. BIT Window.

5.3 Weapons Pull-down Menu

The `Weapons` pull-down menu enables you to load or unload a weapon on the selected weapons station. The available weapons are shown in Table II. When you select this menu, the `Weapons` options (Figure 9) are displayed.

Table II. Available Weapons.

Weapon	Description
Sidearm (AGM-122)	Simulates an AGM-122 Sidearm missile on a LAU-7/A launcher for the selected station(s).
Bomb	Simulates a single bomb on a parent rack for the selected station(s).
Fuel Tank	Simulates a fuel tank loaded on a parent rack for the selected station.
Rocket	Simulates a single rocket pod on a parent rack for the selected station(s).
Sidewinder (AIM-9)	Simulates an AIM-9 Sidewinder missile on a LAU-7/A launcher for the selected station(s).
ECM Pod	Simulates an electronic countermeasures pod on a parent rack for the selected station(s).
TACT Pod	Simulates a Tactical Air Combat Trainer loaded on the selected station(s).
Dispenser	Simulates a single dispenser on a parent rack for the selected station(s)
ITER with bombs	Simulates 0-3 bombs loaded per position on improved triple ejector rack for selected station(s).
ITER with rockets	Simulates 0-3 rockets loaded per position on improved triple ejector rack for selected station(s).
Maverick-E (AGM-65E)	Simulates an AGM-65E Maverick missile on a LAU-117/A launcher for the selected station(s).
Maverick-F (AGM-65F)	Simulates an AGM-65F Maverick missile on a LAU-117/A launcher for the selected station(s).

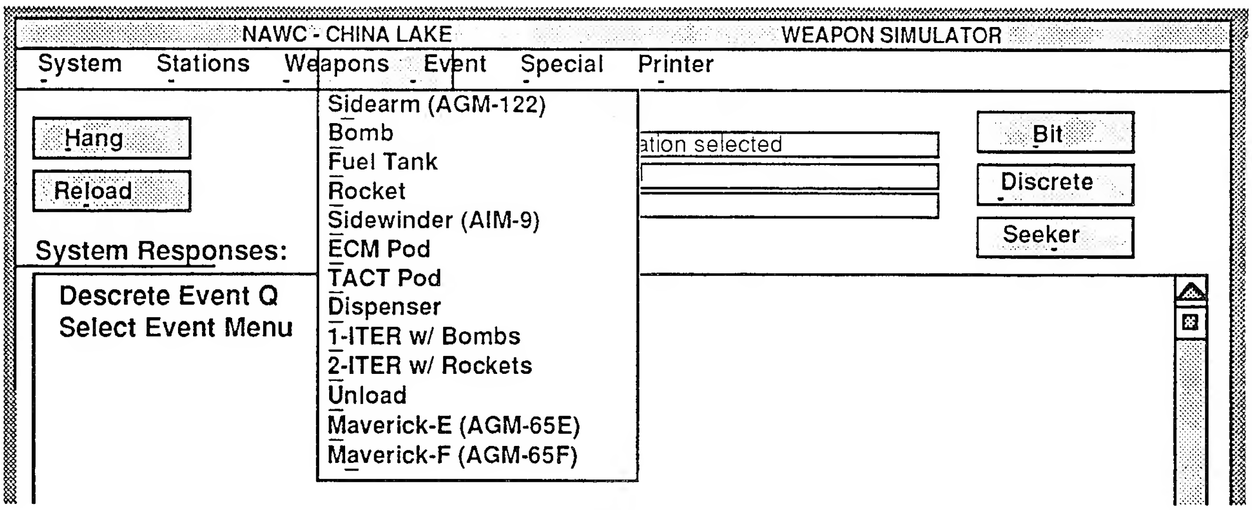


Figure 9. Weapons Pull-Down Menu.

With selection of the weapon options, the system identifies the selected weapon in the Current Weapon field and displays Success: Weapon selection complete. If you select either of the ITER options, the WPNSIM program overlays the ITER Load window (Figure 10).

The three position fields in the ITER Load window correspond to the three positions on a real ITER. When the designation codes are entered and loaded the system displays: **Success: ITER selection complete ...**

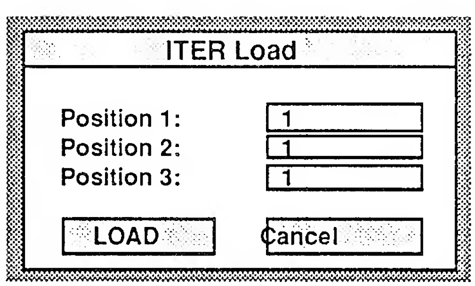


Figure 10. ITER Load Window.

If the Unload option is selected the system displays EMPTY in the Current Weapon field.

5.4 Event Pull-down Menu

The Event pull-down menu produces a report of all events occurring with the selected weapon station(s). An event occurs when a monitored discrete line changes state since the last cycle it was checked, The WPNSIM program generates a system interrupt and the event is recorded.

The event reports are used to verify the SMC and Mission Computer OFPs (example: ensure bombs fell in the correct order and interval for current OFPs).

5.5 Special Pull-down Menu

The Special pull-down menu has two options. Option 1 accesses the Signal Mux menu. Option 2 loads the fuselage gun. The Signal Mux option enables you to select a signal by name from the signal multiplexer boards. The Fuselage Gun option simulates a 25 millimeter fuselage gun pack.

5.5.1 Signal Mux Option

Each signal multiplexer board provides access to that signal through the BNC connector on the front panel of the system. Selection of the Signal Mux option causes the Signal Mux window to be displayed (Figure 11).

The names used by the Signal Mux window for each signal depends on the station to which each board is physically attached. The various mappings of names to signals are provided via the Configuration menu bar at the top of the window. Each Mux pull-down menu has the seven options described in Table III.

Table III. Mux Pull-down Menu Options.

Option	Description	
Station List	Displays signals from the loaded stations.	
Gun List	Displays fuselage gun signals from station 8.	
Raw Data	Displays each signal relay mapped to a register and bit.	
Output 1	BNC Jack 1	When in these windows, selected signals will be switched to these jacks.
Output 2	BNC Jack 2	
Output 3	BNC Jack 3	
Output 4	BNC Jack 4	

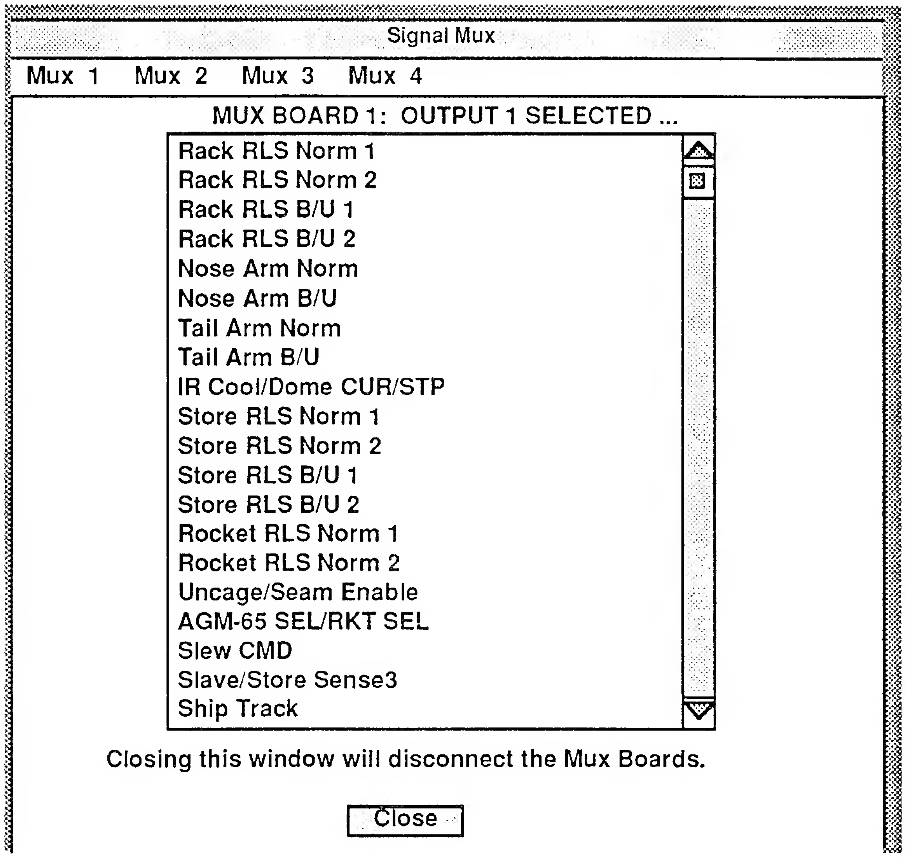


Figure 11. Signal Mux Window.

5.5.2 Fuselage Gun Option

The Fuselage Gun option simulates a 25 millimeter fuselage gun pack. When selected the Fuselage Gun window (Figure 12) overlays the WPNSIM window. The Fuselage Gun window has a **Failures pull-down menu** that provides for failure options; a **Rounds field** that defines the number of gun rounds; a **Load button** that commands the gun board to initialize and start the Gun model; an **Unload button** that commands the gun board to terminate the model and go idle; a **Reload button** that commands the gun board to reinitialize with the original number of rounds specified; and a **BIT button** that causes the gun board to perform BIT and report the results.

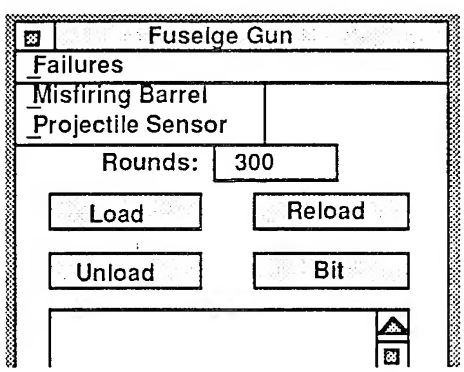


Figure 12. Fuselage Gun Window.

DEMONSTRATION OF INTEGRATED INS + DIFFERENTIALLY CORRECTED GPS DATA PROCESSING ON THE F/A-18

Dr. Karyn Haaland
Nick Albahri
Paul Wilfong*
Ball Corporation, San Diego, California 92121

Mike Hugo
Naval Air Warfare Center, Weapons Division, China Lake, California 93555

ABSTRACT

This paper demonstrates the potential for the use of an inexpensive C/A code receiver, providing differentially corrected GPS data integrated with F/A-18 inertial data, to generate continuous, inexpensive, medium-quality (30-50 ft spherical error) off-range trajectory data, even in the presence of GPS data dropouts. Ball's Advanced Test Data Optimal Processor (ATDOP) is used in an experimental, post-mission configuration to accomplish the following:

- Apply optimal estimation techniques to combine and process differentially corrected GPS solution data with F/A-18 inertial velocity and attitude data

- Provide smooth, continuous solutions across periods of GPS data dropout of up to two minutes.

Results are verified via comparison with cinetheodolite-based reference solutions.

1.0 INTRODUCTION

The TANS GPS receiver is an inexpensive C/A code receiver produced by Trimble Navigation which is readily mounted on the F/A-18. An experimental configuration incorporating a TANS unit has been mounted on aircraft at Naval Air Warfare Center (NAWC), China Lake. This configuration supplies time-tagged GPS solution data to the aircraft 1553 bus for recording and subsequent post-mission processing. There are two issues associated with the use of these data to provide Time-Space-Position Information (TSPI) in the F/A-18 test environment:

- The TANS receiver is an unauthorized C/A code receiver, and thus cannot independently provide position data of the required quality.

- The receiver is subject to stress in a high-dynamic environment. The level of acceleration occasionally exceeds the receiver's specified tolerance. As shown below, the data are consequently subject to substantial dropouts.

The first problem is readily addressed by the use of a GPS base station providing differential corrections for all satellites in view. These corrections accommodate removal of systematic errors due to imperfect ionospheric delay compensation, residual satellite clock errors, and SA effects from the absolute TANS solution. When correctly processed (using off-the-shelf

Trimble PathFinder software in combination with interfaces developed by NAWC personnel), the differential corrections can be used to "correct" TANS receiver solutions to produce position data which is typically good to within 30ft. A representative sample of these results are included in this note.

The second problem poses a significant challenge in the F/A-18 application. Due to the magnitudes and numbers of maneuvers typical for the F/A-18 weapon delivery and Air Combat Training scenarios, GPS data dropouts are common, often during critical time periods. Dropouts are caused by limitations in the satellite tracking loop, as well as antenna masking by the vehicle body. These problems are not readily overcome within the receiver itself without the addition of such aids as an integrated inertial unit and sophisticated satellite switching logic. However, in the post-mission data processing environment, data from the aircraft inertial system can, under many circumstances, be combined by Ball's Advanced Test Data Optimal Processor (ATDOP) to provide continuous medium-quality TSPI, even during GPS data dropouts.

2.0 TECHNICAL APPROACH

Ball's Advanced Test Data Optimal Processor (ATDOP) is a state-of-the-art Square-Root-Information Kalman filter/smoother (SRIF/S) that optimally combines data from diverse TSPI sensors, including laser tracker, radar, theodolite, inertial, doppler, and GPS data. Ball's ATDOP incorporates a vehicle dynamics model to govern the propagation of the state, as well as tracking sensor error models that enable automatic sensor bias calibration. (The term "bias" is used loosely here to refer to a variety of measurement errors including sensor biases, inertial system biases and tilts, GPS clock and bias errors, etc.)

In addition to the standard Kalman filter, which processes data up to and including the estimation time, ATDOP implements a post-mission smoother. Smoothers process information from the entire pass to generate improved state estimates. The smoothing process greatly reduces the sensitivity of the estimator to data dropouts (e.g., due to loss of track by GPS receiver), a phenomenon frequently experienced during the high-dynamics flight profiles that characterize many F/A-18 scenarios. Smoothing, especially in the presence of inertial velocity data, facilitates the calculation of continuous tracking solutions and mitigates effects of the GPS dropouts.

In this demonstration, TANS GPS and F/A-18 inertial data are recorded using the standard aircraft instrumentation system (ALBUS), and a NAWC-developed encoder box tailored to the TANS receiver. The data are reformatted and integrated by the ATDOP, accounting for lever arm, to provide tracking solutions (position, velocity, acceleration, and attitude) referenced to the vehicle INS. The results are compared with cinetheodolite-based reference solutions also referenced to the INS. Results are presented in Section 4 below.

3.0 SCENARIO DESCRIPTION

F/A-18 Flight 108/1937 at NAWC China Lake (carried out January 12, 1995) is selected for the demonstration because both TANS and INS data are recorded and additionally, Askania cinetheodolite data are available for use as a reference or "truth" solutions. The approach is consequently to use the cinetheodolite data as a basis for forming a reference solution to be used to evaluate the integrated TANS/INS solution.

3.1 Overview of Data

Five segments during which INS and TANS data were available are selected for processing (Table 3-1). The aircraft recorder had been turned off between passes, so the length of these segments is somewhat limited. In an operational scenario, the recorder would be left on during regions where TSPI data were required.

Table 3-1. TANS vs. INS Data Availability

Segment	TANS Data Begin Time	TANS Data End Time	INS Data Begin Time	INS Data End Time	Segment Length (sec)
1	80345.0	80441.0	80344.0	80443.0	99
**	81381.0	81398.0	81330.0	81400.0	19
**	N/A	N/A	81524.0	81554.0	24
2	81917.0	81992.0	81917.0	81992.0	75
3	82142.0	82326.0	82142.0	82325.0	183
4	82367.0	82480.0	82369.0	82481.0	112
5	82542.0	82638.0	82543.0	82601.0	96

** These segments are not processed for this study due to their short duration

For the purpose of demonstration, regions exhibiting the following characteristics are of particular interest:

- Average (8-15 sec) to substantial (>20 sec) TANS data dropouts followed by a resumption of TANS data availability.
- Continuous INS data availability before, during and after TANS data dropouts.
- Unique maneuvers with stressful dynamic characteristics.

The TANS data segments selected are summarized in Table 3-2. These segments include periods of stressful (up to 5G) turn, roll, and dive maneuvers. Dynamic profiles for the individual segments are included in the Section 4 - Results.

In addition, in order to study ATDOP performance in the presence of larger TANS data gaps, some TANS data gaps are artificially extended by deleting data points. Table 3-2 lists in units of seconds for each segment the processing begin and end time (UTC seconds of day), the length of the segment, the largest data gap encountered and any artificially extended data gaps.

Table 3-2. ATDOP TANS Solution Data Processing

Segment	Begin Time	End Time	Duration	Largest Gap	Artificially Extended Gaps
1	80345.0	80441.0	96.0	9.0	0
2	81958.0	81992.0	34.0	14.0	0
3	82145.0	82326.0	181.0	25.0	120.0
4	82370.0	82480.0	110.0	14.0	0
5	82544.0	82638.0	94.0	19.0	30.0

3.2 ATDOP Processing

The INS data is available and processed at 20 Hz; TANS data is available at 1 Hz. The ATDOP filter/smoother is executed at 10 Hz.

The ATDOP filter is tuned by performing multiple executions of the filter and iteratively adjusting tuning parameters until the filter residuals exhibit approximately "white", zero mean behavior. Tuning parameters include measurement noise, measurement bias statistics, INS error statistics, a vehicle dynamics model, and GPS track point. The GPS and INS residuals are included in the Section 4.2 Results for reference. A visual examination indicates that the residuals are essentially zero mean, unstructured (white) and lie within their one-sigma uncertainties more than 67% of the time. This nearly text-book behavior verifies the filter tuning and is a strong indication that the smoother solutions are of the quality indicated by the one-sigma uncertainties.

The TANS solution measurement errors are characterized by a 5 ft white noise plus a Markov process error with an RMS value of 25 ft and time constant of 10 seconds. The Markov error is included to model the effects of S/A, multipath, interchannel bias, residual ionospheric errors, etc. The filter INS model is tuned based on previous experience with the F/A-18 INS (a 3 state error model is used to model INS errors; this could be expanded to up to 15 states if enough data were available to admit observability of the additional states). For each of the data segments, a vehicle dynamics model is constructed corresponding to the dynamics of the flight during that segment.

The final, "best" ATDOP results are transformed to the cine-based reference track point and used to evaluate the ATDOP TANS + INS solutions. These comparisons are presented in Sections 4.2.x, where x indicates the segment number. Uncertainties in the ATDOP solution are also included in these sections as an indicator of the quality of the ATDOP solution. The statistical rule is that under conditions of perfect modeling, the error in the ATDOP solution will be less than the indicated uncertainty 66.6% of the time. Note that these plots do not account for the uncertainty in the reference solution; however, the error is assumed to be dominated by the errors in the TANS solution. The variations in the one-sigma values that are observed in the plots are due to a combination of filter tuning effects (process noise on acceleration is increased during periods of acceleration), and due to data availability as the TANS drops in and out. In general, the more data, the less stressful the dynamics, and the better the geometry, the smaller the uncertainties.

4.0 RESULTS

This section presents results, including raw TANS and ATDOP processed TANS plus INS. Solutions are compared to the cine-based reference solutions.

4.1 Differential TANS Solutions - Without ATDOP Processing

This section summarizes the differential TANS data for Flight 1937. Figure 4.1-1 summarizes the TANS solution points in the form of plots of differences between TANS data and the cine-based reference solutions. The figure shows segment 3 results as a sample of solution results which are typical of the entire test. Compensation for track point differences is included.

As shown in the figure, the GPS-based solution data from the experimental configuration exhibit two unsatisfactory phenomena:

- Data dropouts during maneuvers and at other unexplained times (short dropouts are not evident on the plot)
- Several "bad" data points just prior to a dropout.

The ATDOP processor is used in this study to incorporate INS data and to smooth across data dropouts, thus mitigating the effects of the first problem. These results are presented in Section 4.2.

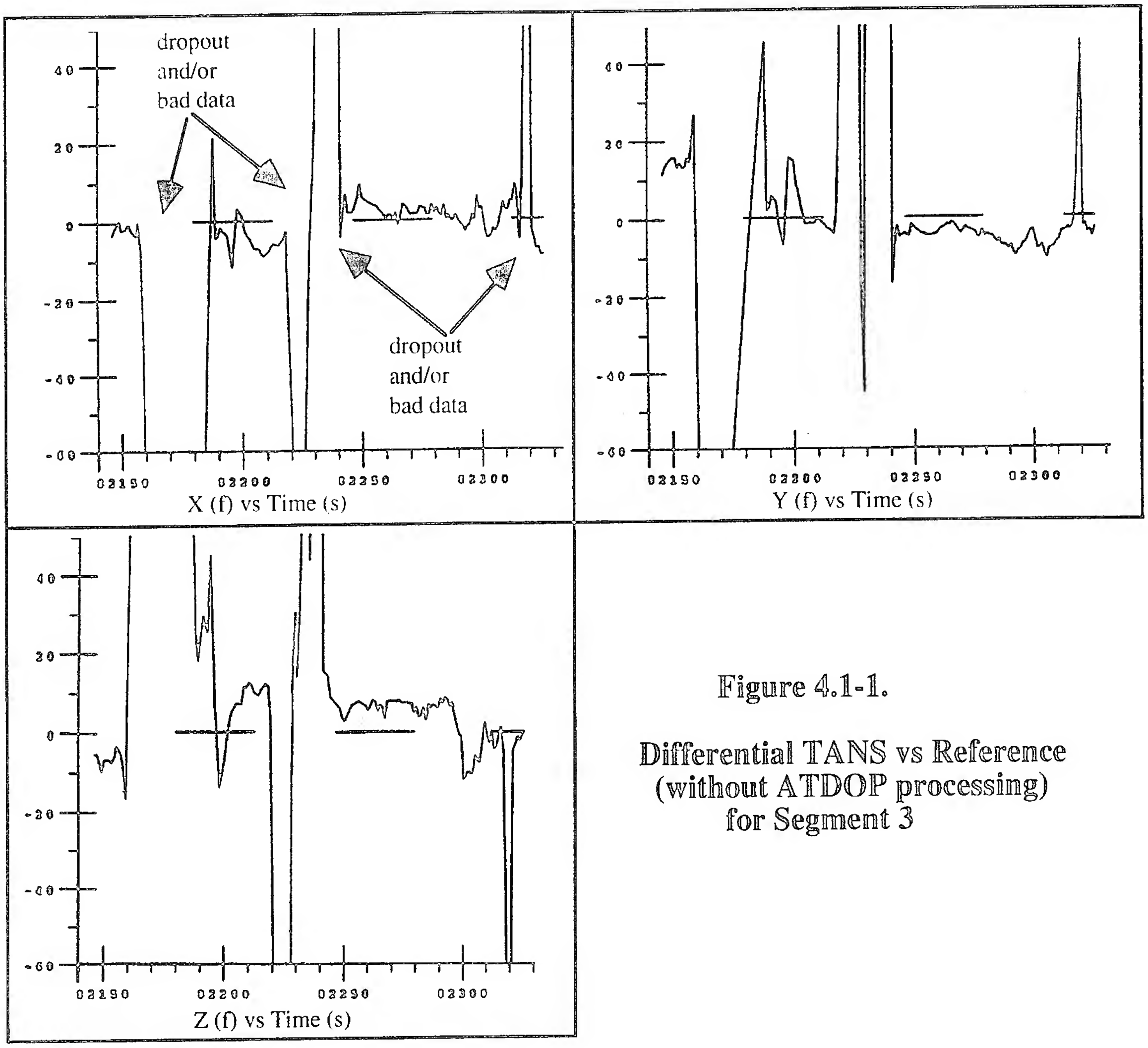


Figure 4.1-1.

Differential TANS vs Reference
(without ATDOP processing)
for Segment 3

The second problem poses more of a challenge to ATDOP. Some of the "bad" data points are automatically rejected by ATDOP during preprocessing. In particular, ATDOP incorporates bounds and first difference checks which screen "very bad" data at an early stage. Additionally, data which are not bad enough to trigger these crude validity checks are often screened and eliminated during the course of filtering by statistical residual checks. These checks compare the residual, that is, the difference between the measurement and the "predicted measurement" based on the current estimate of the state, with the Kalman Filter statistics for the residual. Those measurements that exceed their statistics by a tunable value are not incorporated by the filter and are thus prevented from corrupting the solution. However, for the current case in which the "bad" data are not too bad, i.e., they appear reasonable and are difficult to automatically distinguish from good data, ATDOP has trouble. Some of these points thus had to be eliminated manually. Additional tuning of ATDOP would be required to productionize this process.

4.2 Differential TANS plus INS Solutions - ATDOP

Results of the ATDOP processing of both differential TANS and INS solutions are presented in this section. This section summarizes the results and analysis for the five segments that are studied.

4.2.1 Segment One

The flight profile for this segment is shown in Figure 4.2.1-1a (acceleration vs. time) and 4.2.1-1b (plan view and up vs. time), and exhibits a left turn, 4G maneuver.

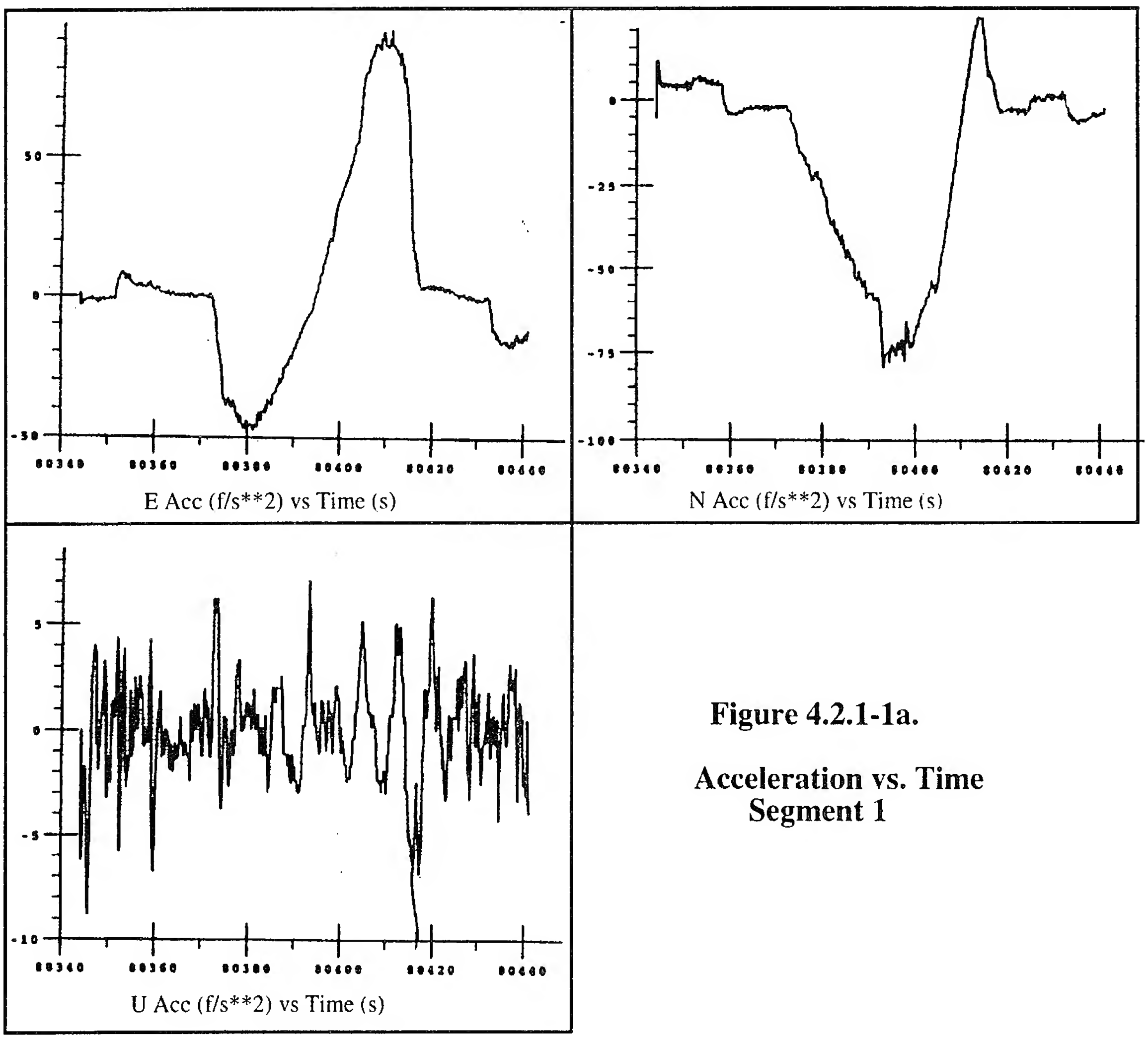
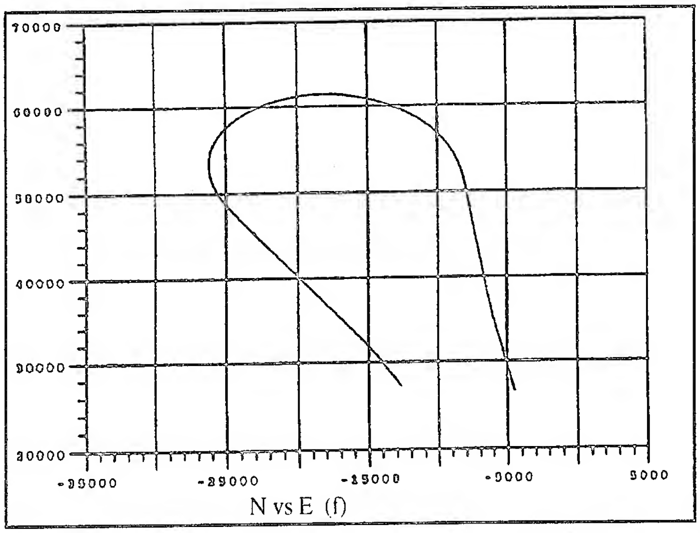


Figure 4.2.1-1a.

Acceleration vs. Time
Segment 1



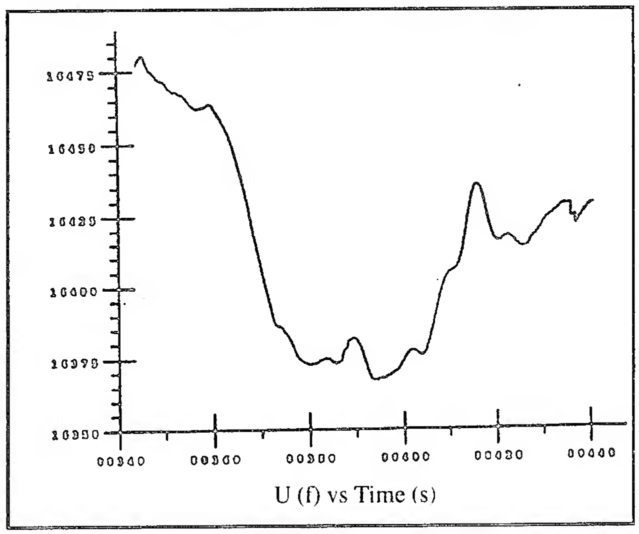


Figure 4.2.1-1b.

Position
Segment 1

<u>Solutions:</u> The actual ATDOP filter errors (TANS+INS vs. cine-basaed reference solution) are shown as the central line in Figure 4.2.1-2. The two outer curves indicate one-sigma predicted uncertainties in the ATDOP filter solution. Smoothed versions of the ATDOP TANS/INS vs. cine-based reference solution are shown in Figure 4.2.1-3.

For this segment, there are two moderate TANS data gaps. The first data gap is at time 80380 and lasts for 8 seconds; the second gap is at time 80413 and lasts for 9 seconds. ATDOP continues to filter inertial data during these periods. Note that for both gaps, ATDOP identifies a few of the TANS measurements before data dropout as "bad" measurements (indicated by asterisks on the plot), and consequently refuses to incorporate them in the solution. Observe that even though there are maneuvers during the dropout, the errors remain bounded by less than 30 feet throughout.

The benefit of smoothing is evident via a comparison of Figures 4.2.1-2 and 4.2.1-3. Figure 4.2.1-2 illustrates how upon resumption of the TANS data after a gap the error growth due to the free inertial operation is quickly corrected. During smoothing, this "correction" is effectively incorporated in the solution for the entire segment (i.e., including the dropout). Evaluation of the uncertainties confirm this performance statistically. ATDOP filter predicted uncertainties in the TANS/INS solution grow during dropouts, but very quickly return to 20 feet upon resumption of

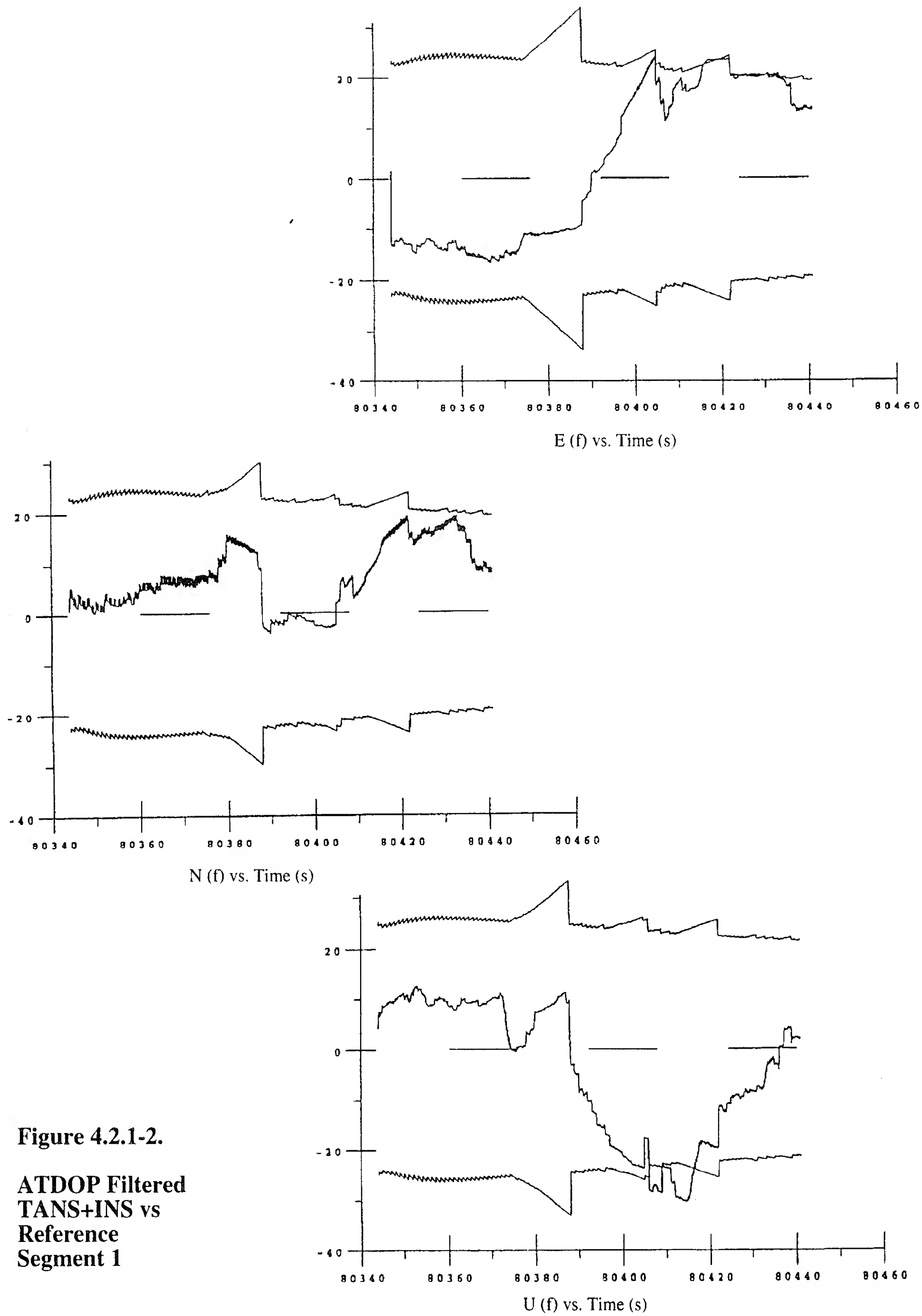


Figure 4.2.1-2.

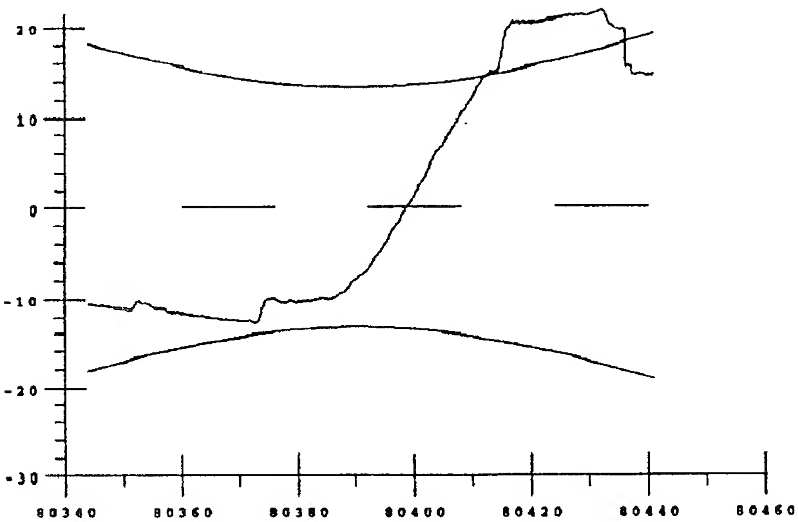
ATDOP Filtered TANS+INS vs Reference Segment 1

TANS solutions. Additionally, the smoothed uncertainties (and similarly the errors) are reduced during the dropout as well.

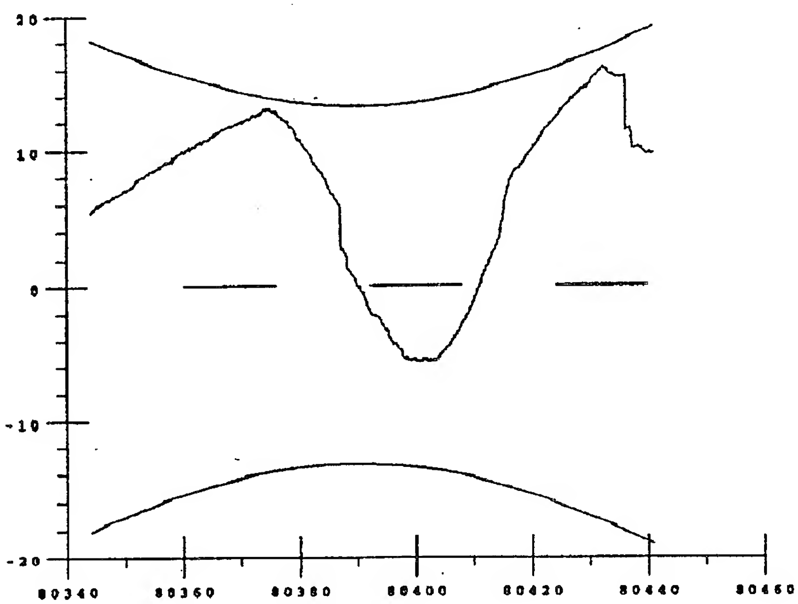
Comparable results are presented for additional segments in the sections which follow.

Residuals: Residuals are the difference between the measurement and the filter "predicted" measurement, i.e., the measurement which would result if the filter a priori state were exact and there were no unmodeled measurement errors.

Just prior to the TANS dropout, the TANS solutions degrade rapidly and are automatically rejected by TDOP, and thus are not allowed to contaminate the solution. This is critical to the success of the filter under conditions where the TANS solutions degrade rapidly prior to a dropout. The rejection of this "bad" data is evidenced by the TANS residual plots shown in Figure 4.2.1-4; when the discrepancy exceeds the predicted accuracy (shown by the plus and minus one-sigma values) by greater than a user settable threshold (currently set from 1 to 1.5), the measurements are automatically rejected. INS velocity filter residuals are shown in Figure 4.2.1-5. Note that these are approximately white and lie within their predicted one-sigma values, thus indicating reasonable filter tuning. Note also that the TANS residual uncertainies in Figure 4.2.1-4 are inflated relative to the actual residual values, which reflects an inflated TANS measurement model ito account for unmodelled bias errors in the data.



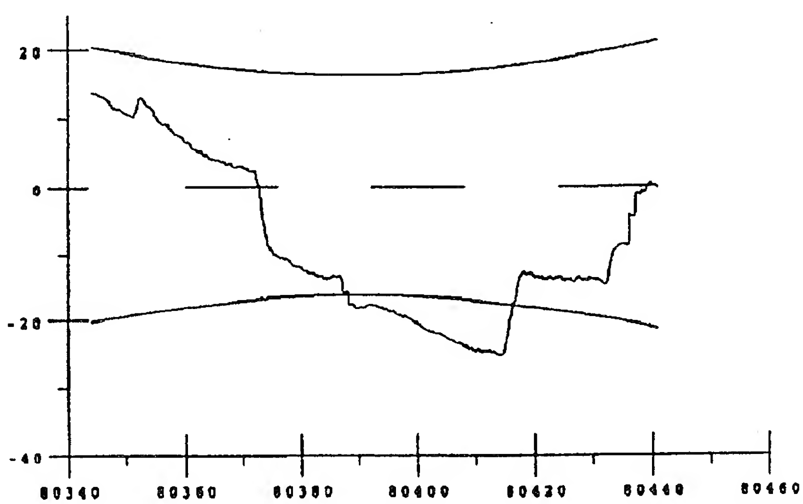
E (f) vs. Time (s)



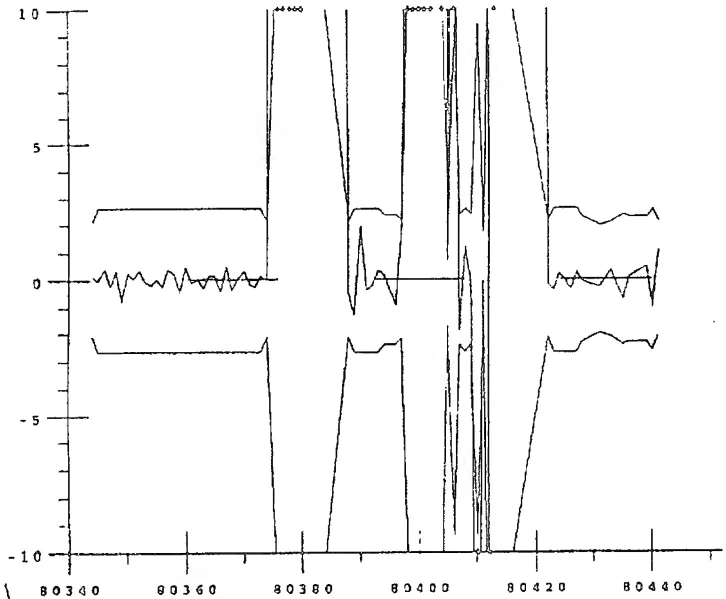
N (f) vs. Time (s)

Figure 4.2.1-3.

ATDOP Smoothed TANS+INS vs Reference Segment 1

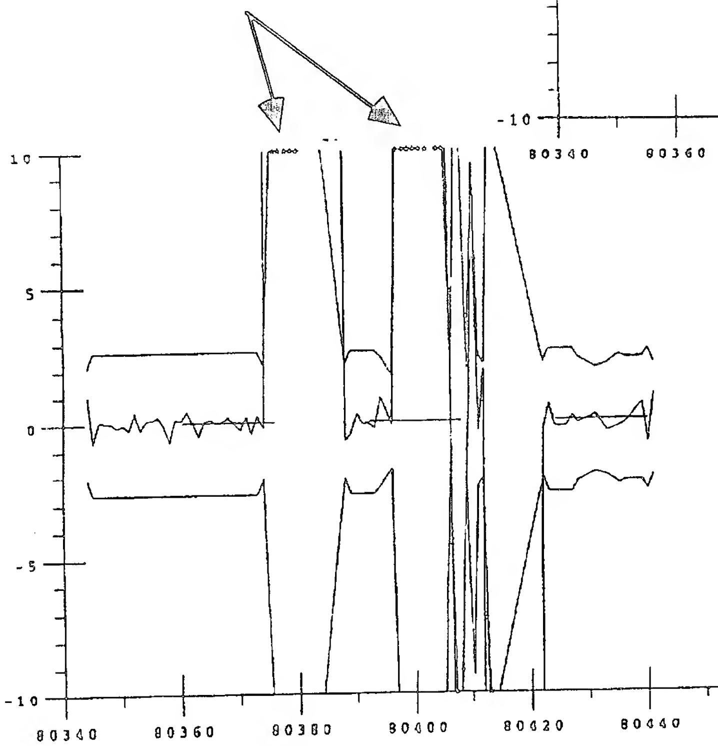


U (f) vs. Time (s)



X (f) vs. Time (s)

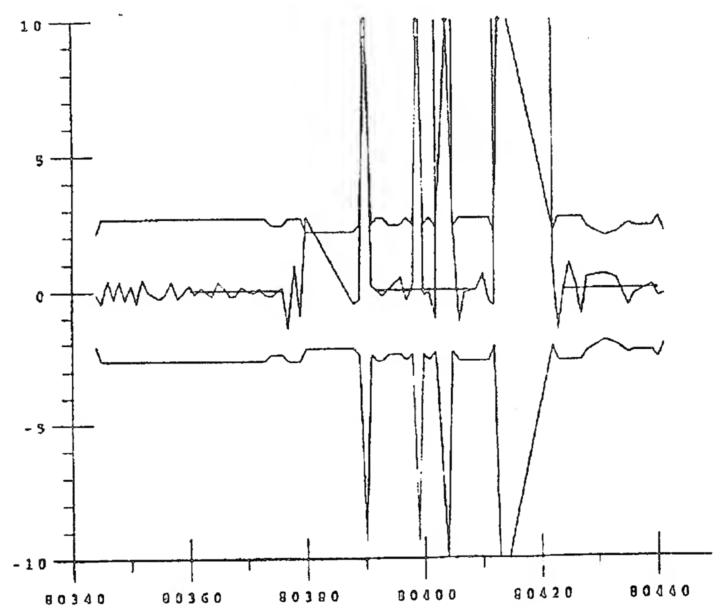
Note: asterisks (*) indicate rejected measurements



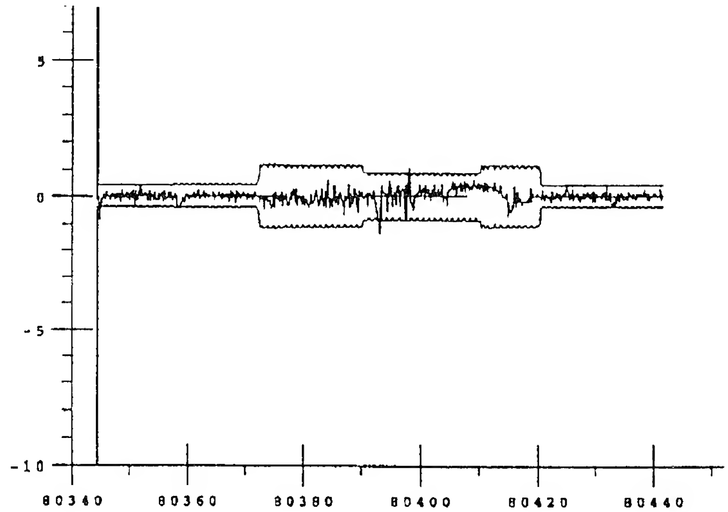
Y (f) vs. Time (s)

Figure 4.2.1-4.

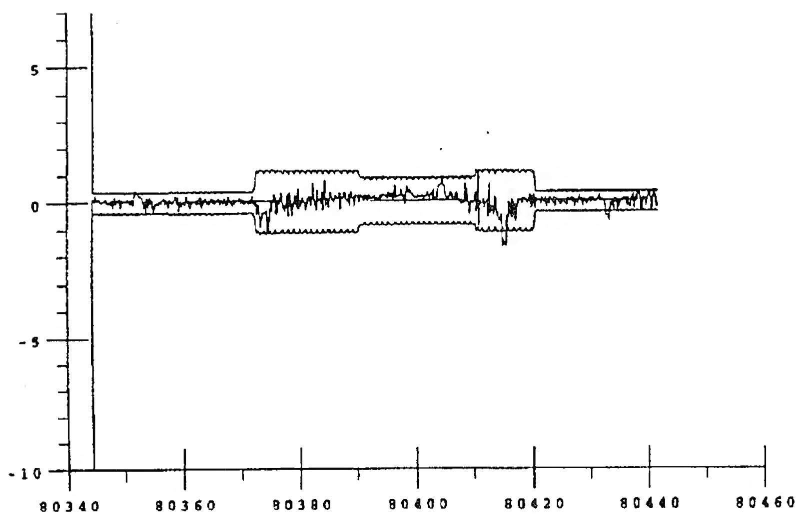
TANS Residuals Segment 1



Z (f) vs. Time (s)



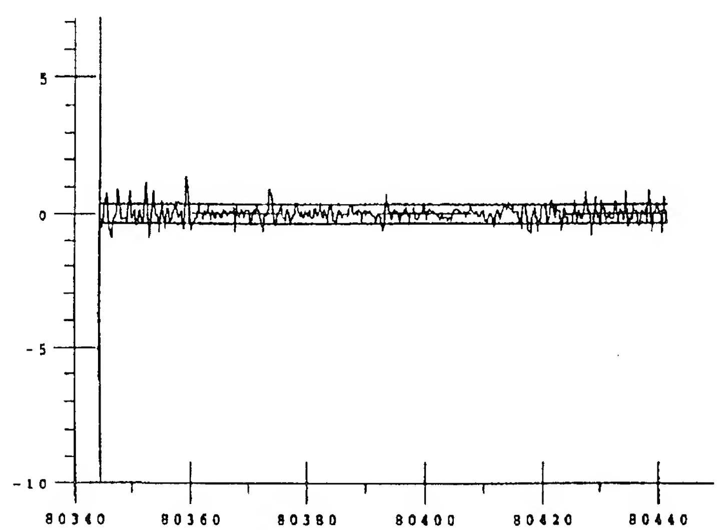
X Vel (f/s) vs. Tims (s)



Y Vel (f/s) vs. Tims (s)

Figure 4.2.1-5.

INS Velocity Residuals Segment 1



Z Vel (f/s) vs. Tims (s)

4.2.2 Segment Two

The flight profile for this segment is shown in Figures 4.2.2-1a and b, and exhibits maneuvers in both horizontal and vertical directions, resulting from a steep climb.

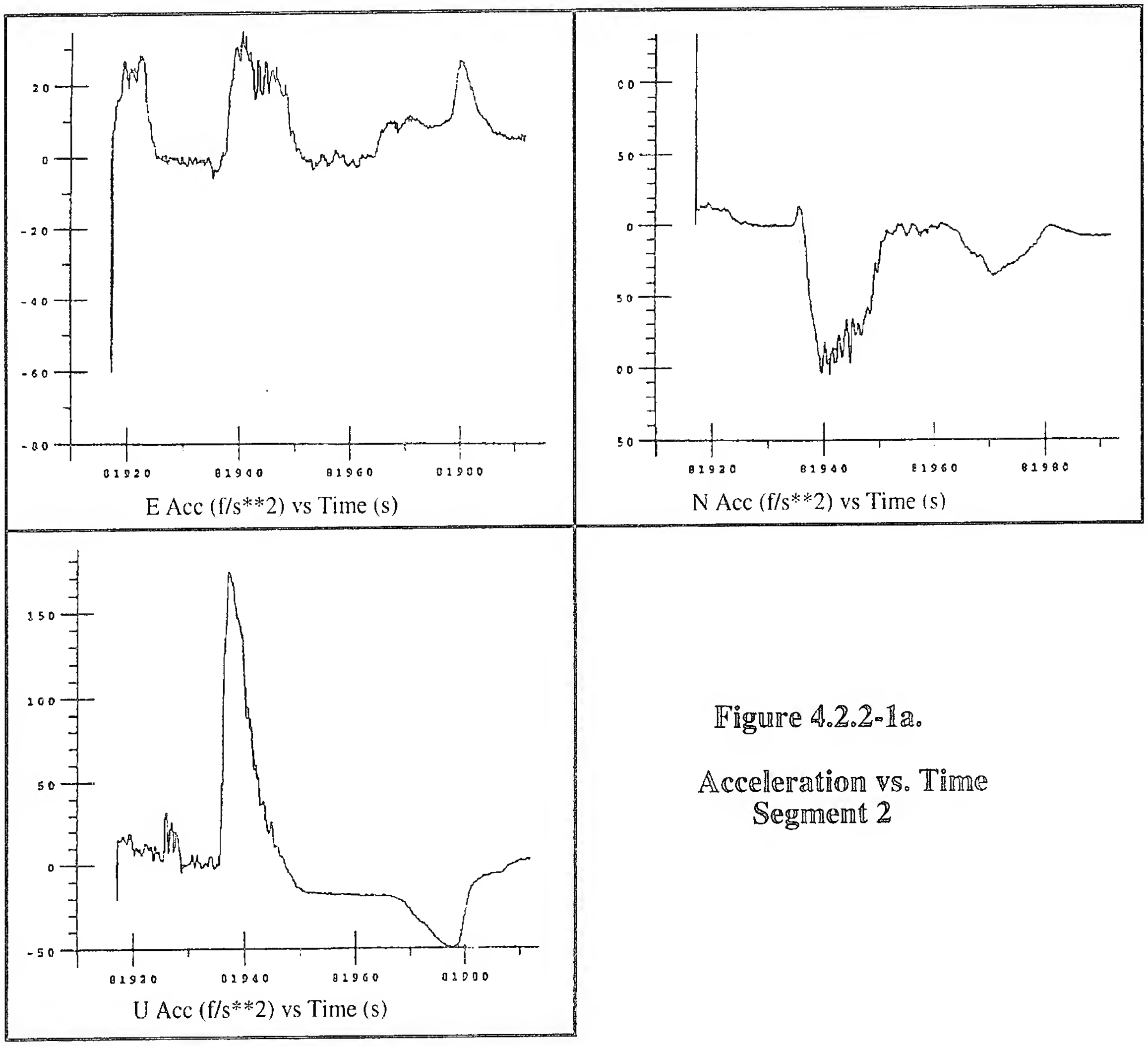


Figure 4.2.2-1a.

Acceleration vs. Time
Segment 2

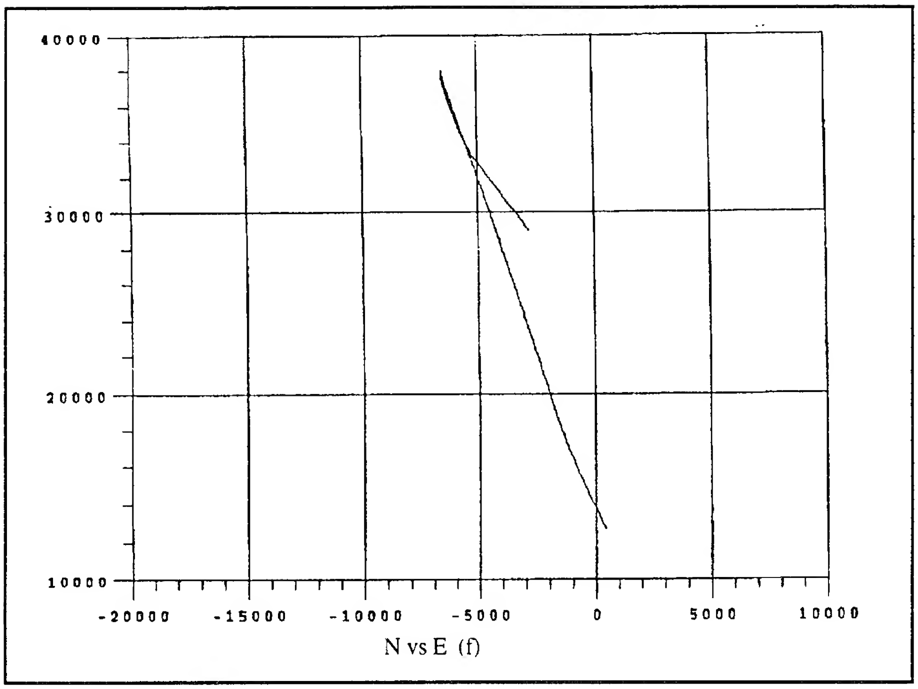
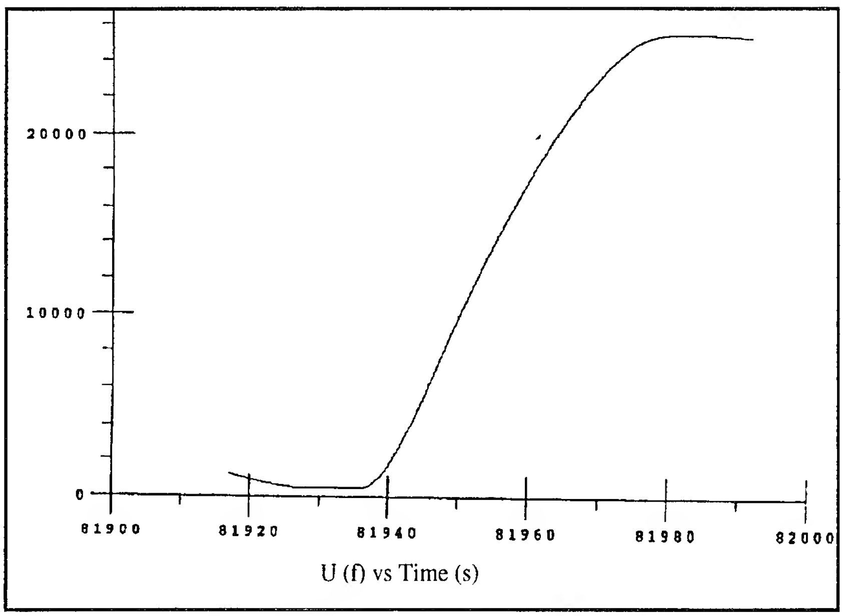


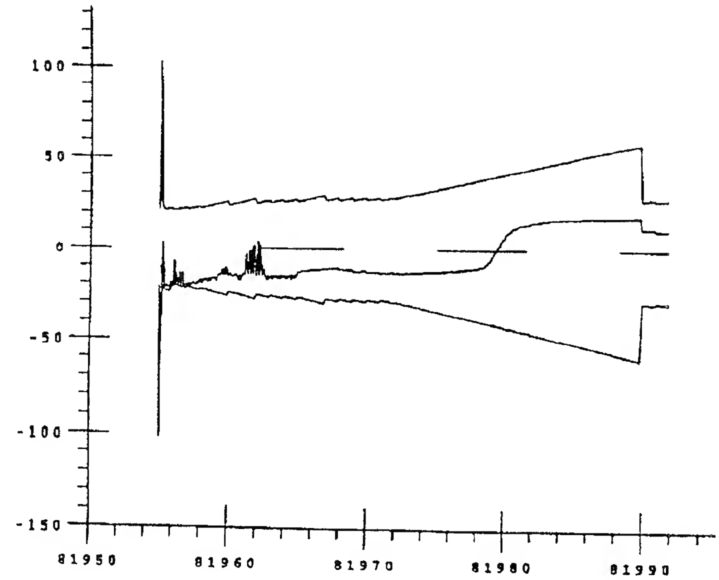
Figure 4.2.2-1b.

Position Segment 2

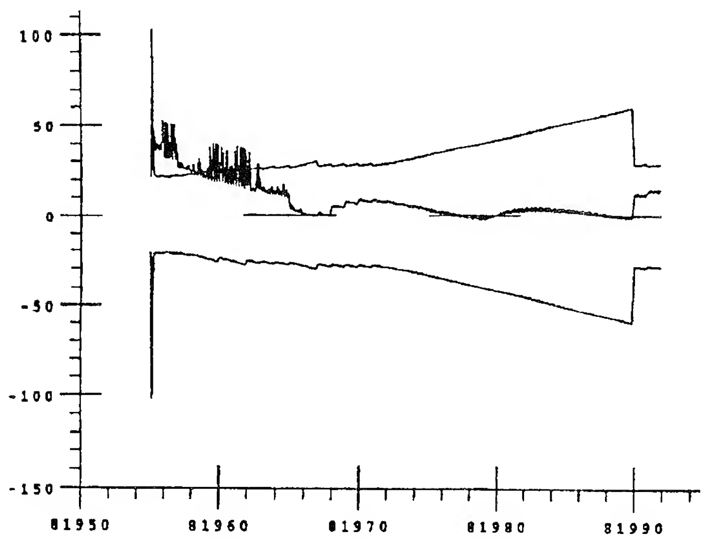


Solutions: The actual errors (TANS+INS vs. cine-based reference solution) are shown in Figure 4.2.2-2. This segment shows successful ATDOP processing (filtering/smoothing) across average data gaps and the advantage gained using differential TANS aided with INS data. The data gap starts at time 81976 and lasts for 14 seconds. Across this data gap, ATDOP is utilizing only inertial data. Note that even though there are maneuvers during the dropout, the errors remain bounded by 30 feet. Smoothed versions of the TANS/INS vs. cine-based results for this case are shown in Figure 4.2.2-3. ATDOP filter predicted uncertainties in the TANS/INS TDOP solution grow during dropouts to 70 feet, but very quickly return to 30 feet upon resumption of TANS solution. Additionally, the smoothed uncertainties (and similarly the errors) are reduced during the dropout as well.

Residuals: Just before the data dropout region, ATDOP flags (rejects) some of the TANS measurements as "bad" measurements. Because of the TANS data rejection, the effective data gap is approximately 20 seconds long. This is typically the case when there is a receiver loss of track usually around maneuvering. Note that during the data dropout period the TANS residual uncertainties grow to reflect the absence of TANS measurements and drop back when TANS data resumes. TANS residual plots are shown in Figure 4.2.2-4. INS velocity filter residuals are shown in Figure 4.2.2-5. Note that these are approximately white and lie within their predicted one-sigma values, thus indicating reasonable filter tuning. The TANS filter residual uncertainties in Figure 4.2.2-4 reflect the inflated measurement model used to account for unmodelled bias errors.



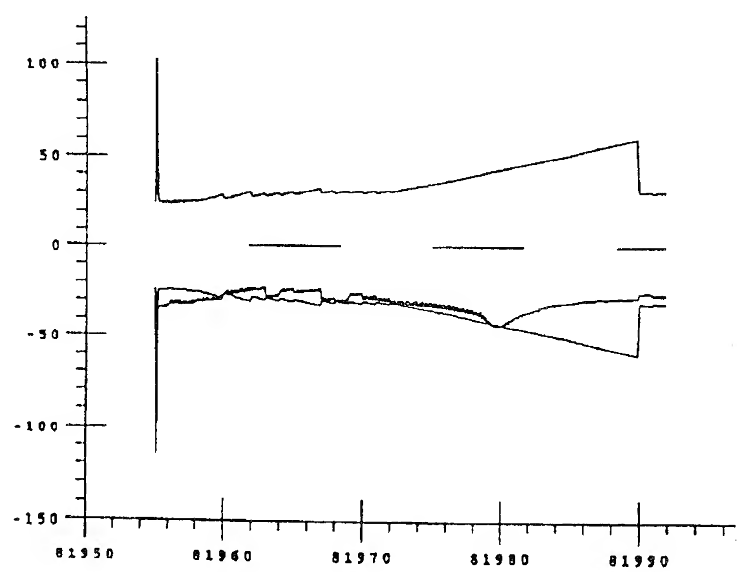
E (f) vs. Time (s)



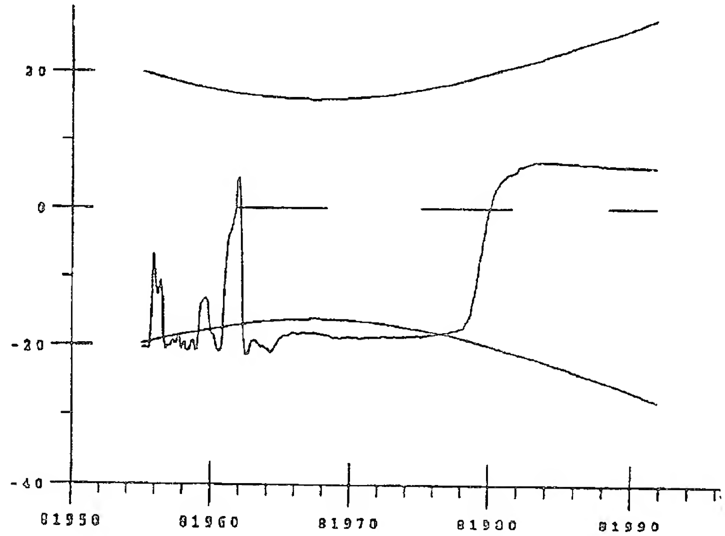
N (f) vs. Time (s)

Figure 4.2.2-2.

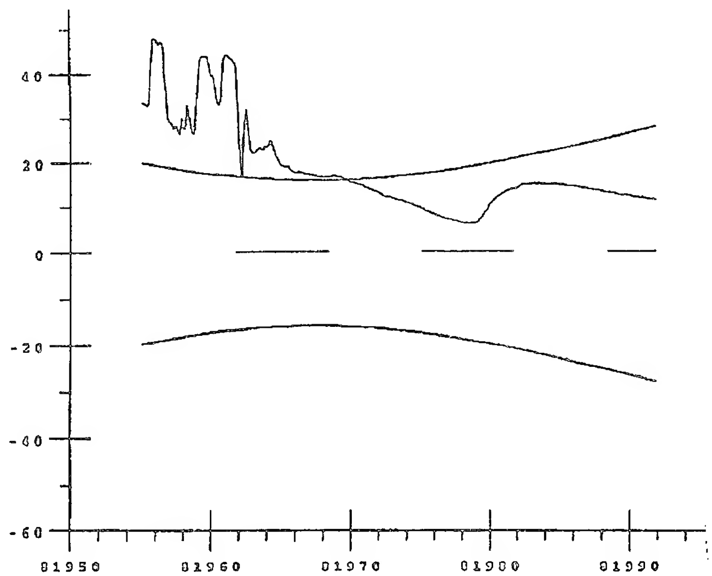
ATDOP Filtered TANS+INS vs Reference Segment 2



U (f) vs. Time (s)



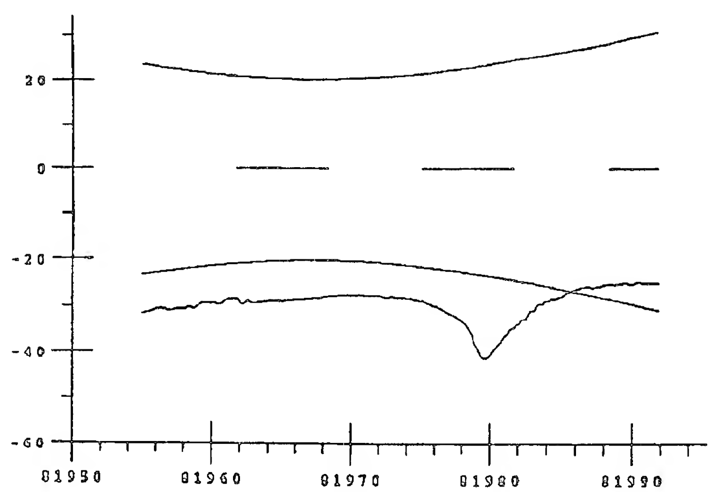
E (f) vs. Time (s)



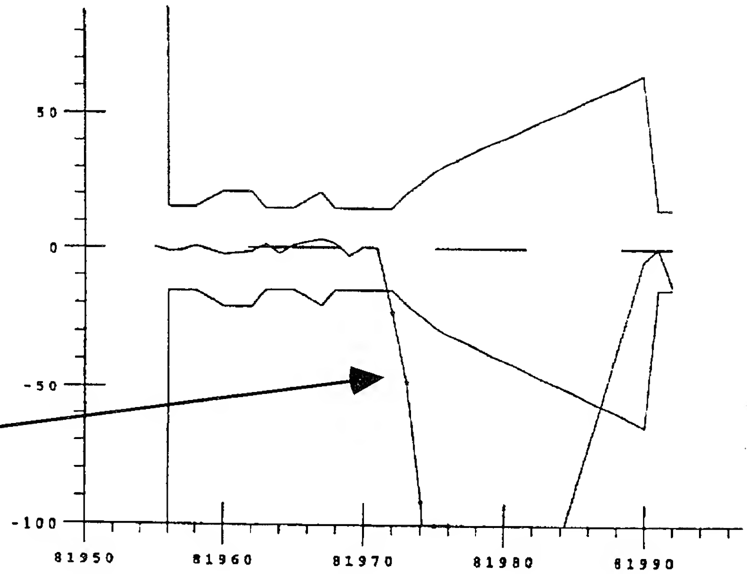
N (f) vs. Time (s)

Figure 4.2.2-3.

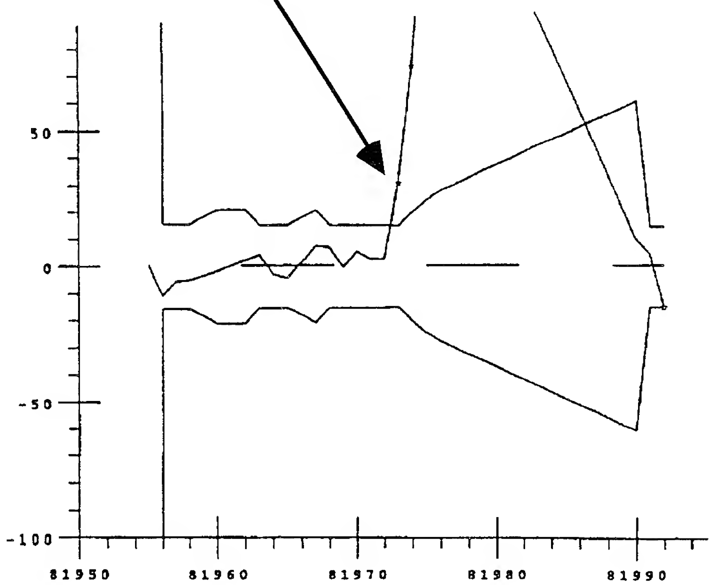
ATDOP Smoothed TANS+INS vs Reference Segment 2



U (f) vs. Time (s)



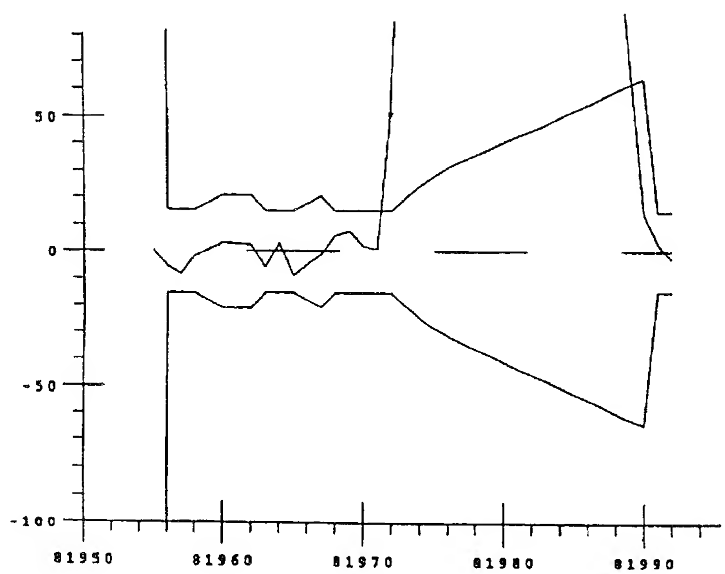
X (f) vs. Time (s)



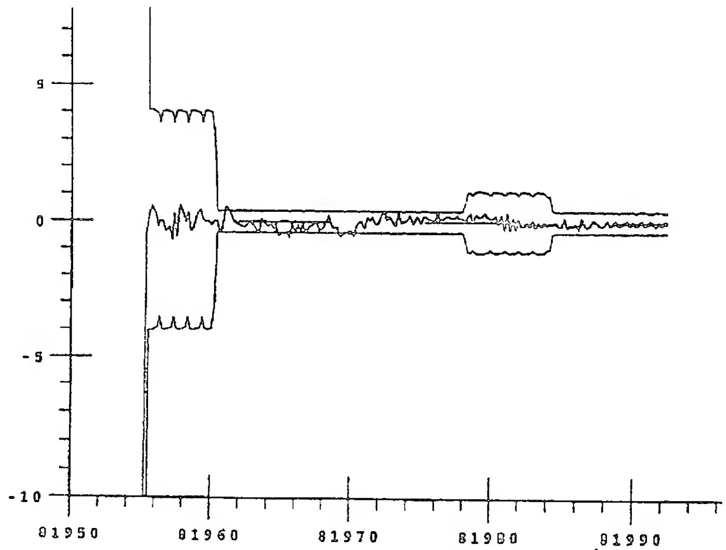
Y (f) vs. Time (s)

Figure 4.2.2-4.

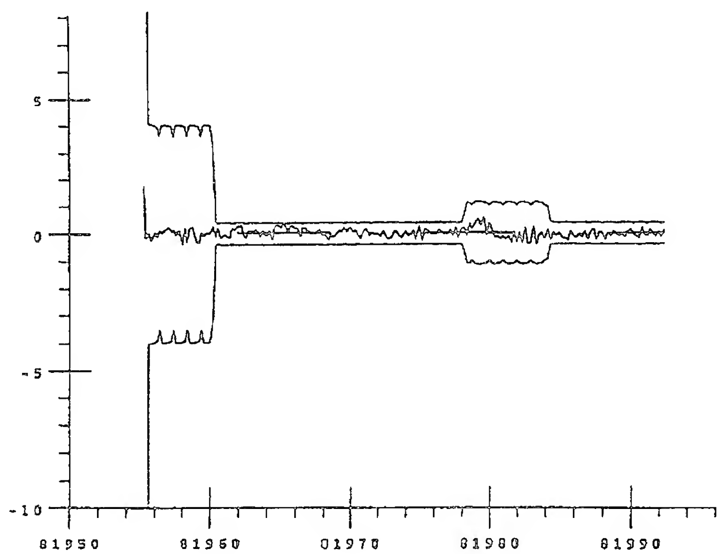
**TANS Residuals
Segment 2**



Z (f) vs. Time (s)



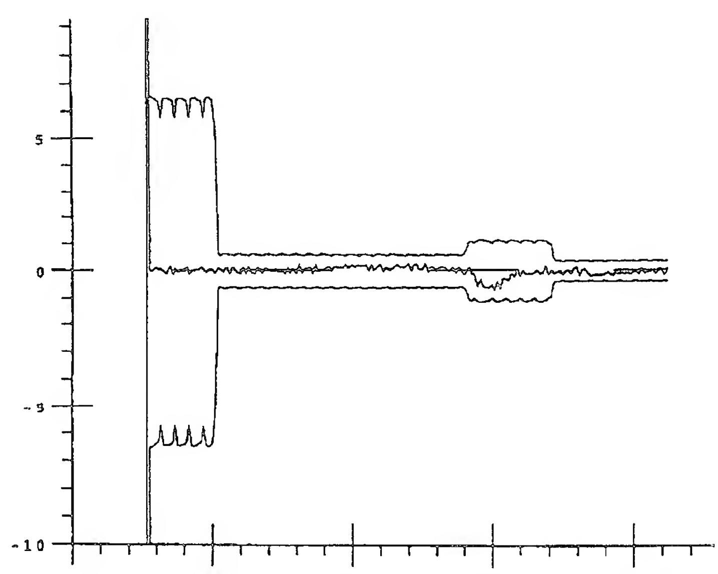
X Vel (f/s) vs. Tims (s)



Y Vel (f/s) vs. Tims (s)

Figure 4.2.2-5.

INS Velocity Residuals
Segment 2



Z Vel (f/s) vs. Tims (s)

4.2.3 Segment Three

The flight profile for this segment is shown in Figures 4.2.3-1a (acceleration vs. time) and 4.2.3-1b (plan view and up vs. time) and exhibits maneuvers in both horizontal and vertical directions, resulting from a right-hand turn and a steep climb.

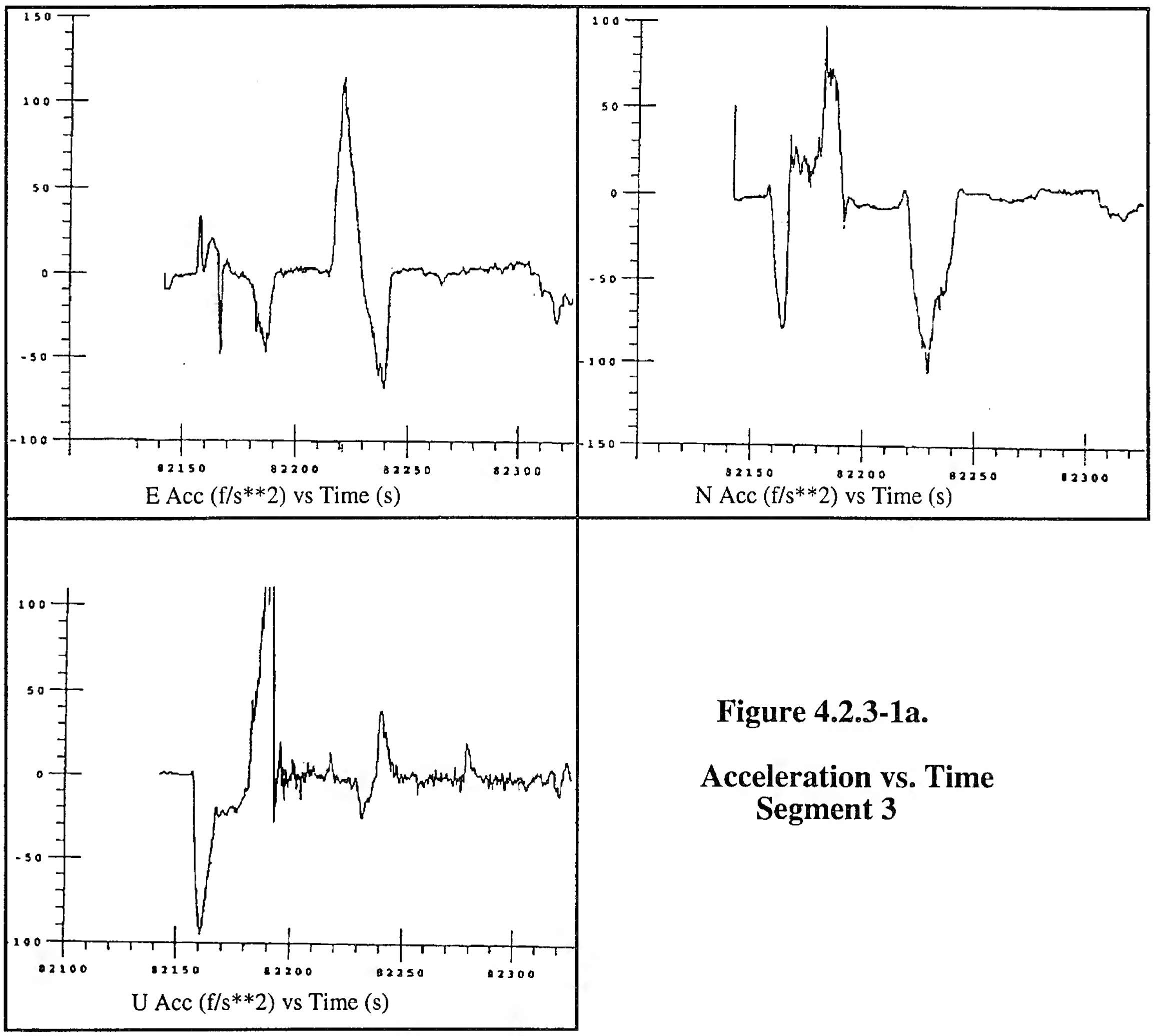
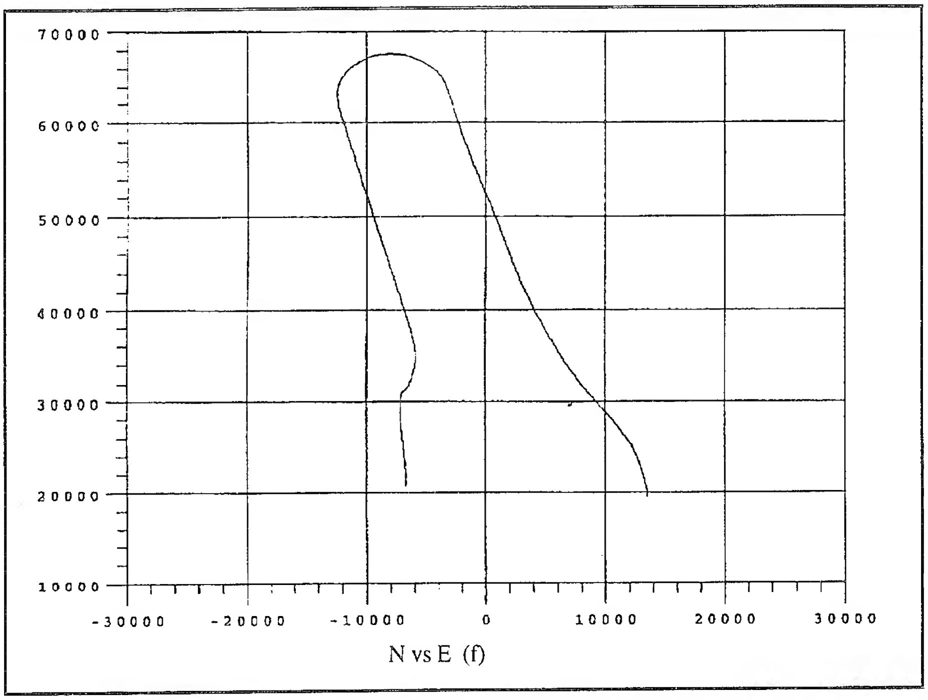


Figure 4.2.3-1a.

Acceleration vs. Time
Segment 3



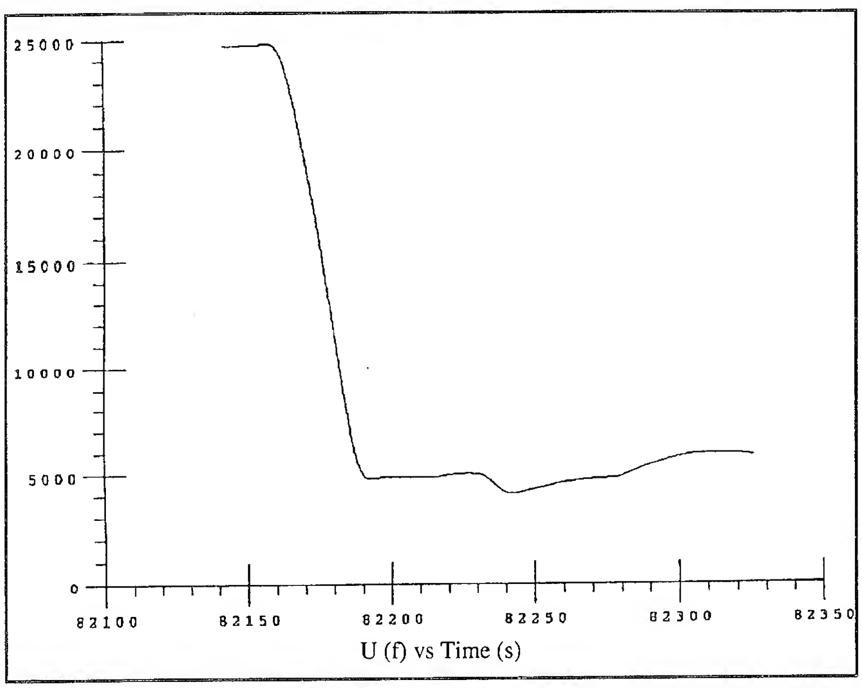
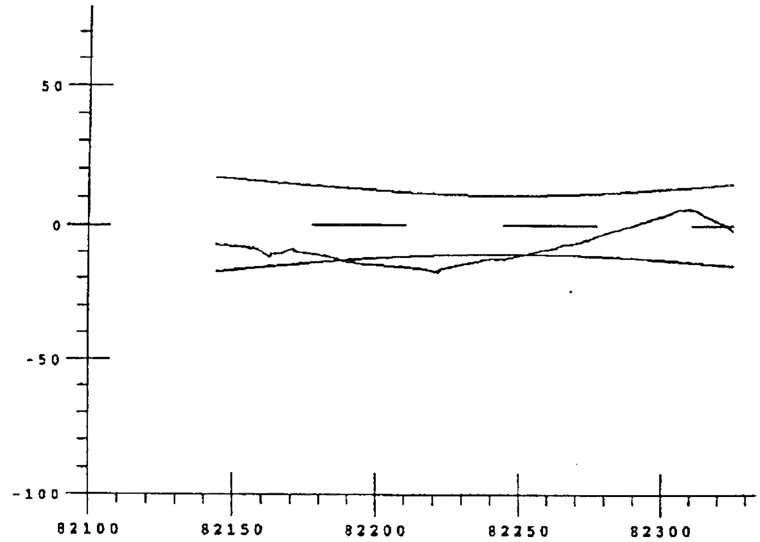


Figure 4.2.3-1b.

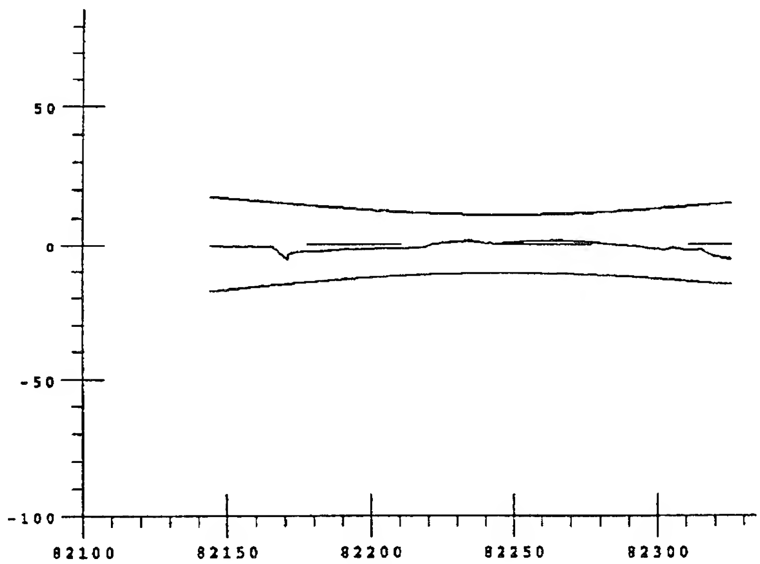
Position
Segment 3

Only the smoother solutions are shown in this and subsequent sections, due to the similarity in filter and residual behavior to the first two segments.

Solutions: The smoother errors (TANS+INS vs. cine-based reference solution) are shown in Figure 4.2.3-2. The purpose of this segment is to show ATDOP processing across substantially larger gaps and the advantage gained using differential TANS aided with INS data. The data gap starts at time 82158 and lasts for 120 sec. During this period, ATDOP is utilizing only inertial data. Note that even though there are maneuvers during the dropout, the errors remain bounded by



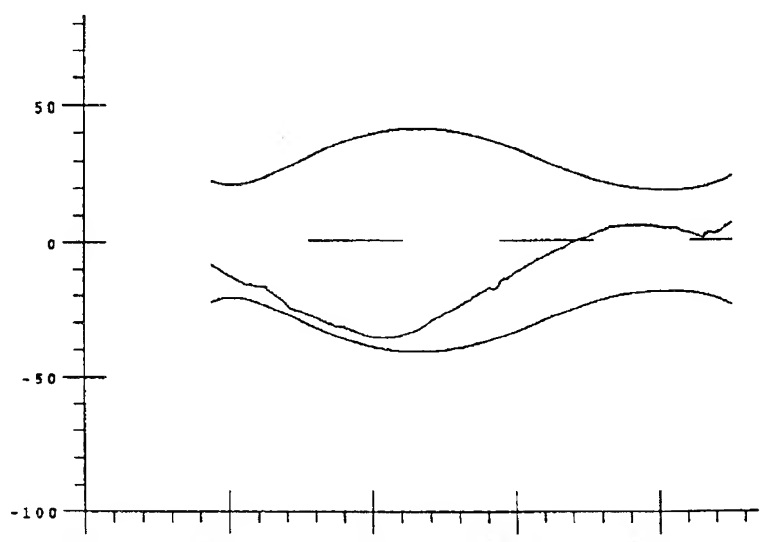
E (f) vs. Time (s)



N (f) vs. Time (s)

Figure 4.2.3-2.

ATDOP Smoothed TANS+INS vs Reference Segment 3



U (f) vs. Time (s)

50 feet. ATDOP filter predicted uncertainty behavior is similar to results shown for the first two segments: TANS/INS TDOP solution uncertainties grow during dropouts but very quickly return to reduced values upon resumption of TANS solutions. Similarly, the smoothed uncertainties (and similarly the errors) are reduced during the dropout as well.

Residuals: TANS/INS residual behavior for Segment 3 are similar to results shown for the first two segments, and are approximately white and lie within their predicted one-sigma values, thus indicating reasonable filter tuning. During the data dropout period the residual uncertainties grow to reflect the absence of TANS measurements and drop back when TANS data resumes. The TANS residual uncertainties reflect the inflated measurement model used to account for unmodelled bias errors.

4.2.4 Segment Four

The flight profile for this segment is shown in Figures 4.2.4-1a (acceleration vs. time) and 4.2.4-1b (plan view and up vs. time) and exhibits maneuvers in both horizontal and vertical directions, resulting from a steep climb.

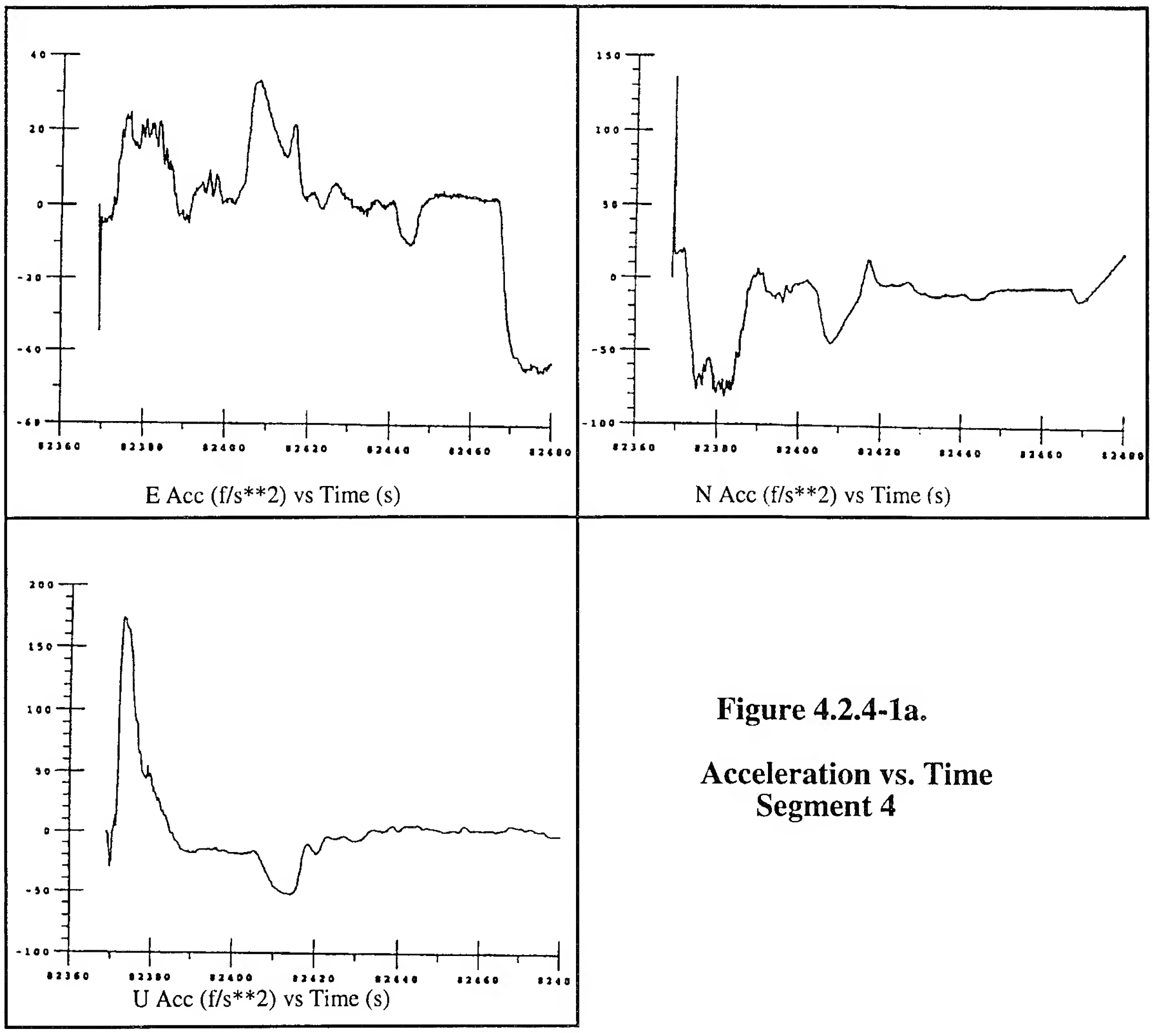
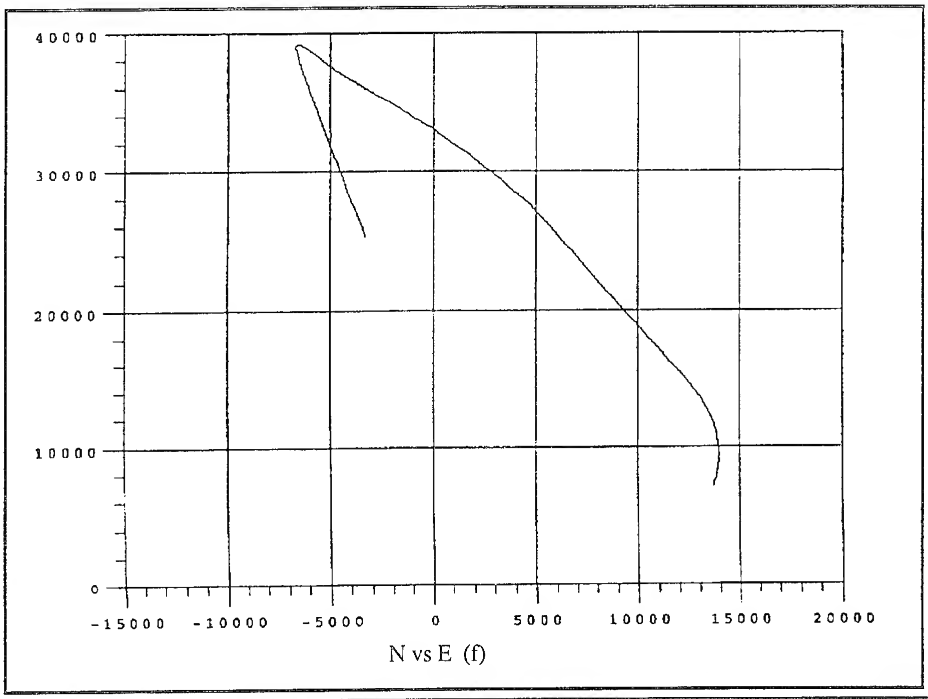


Figure 4.2.4-1a.

Acceleration vs. Time Segment 4



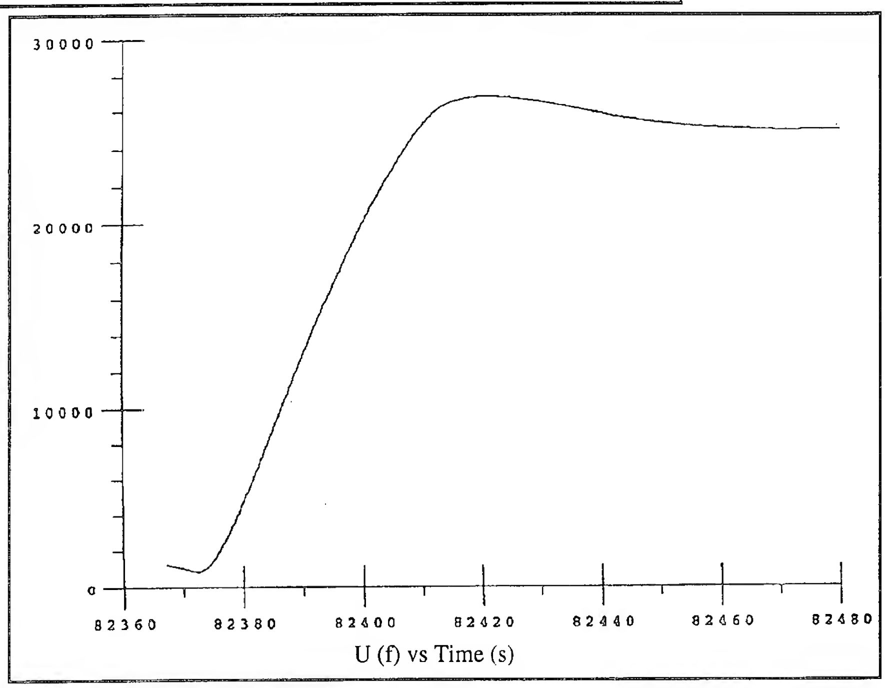


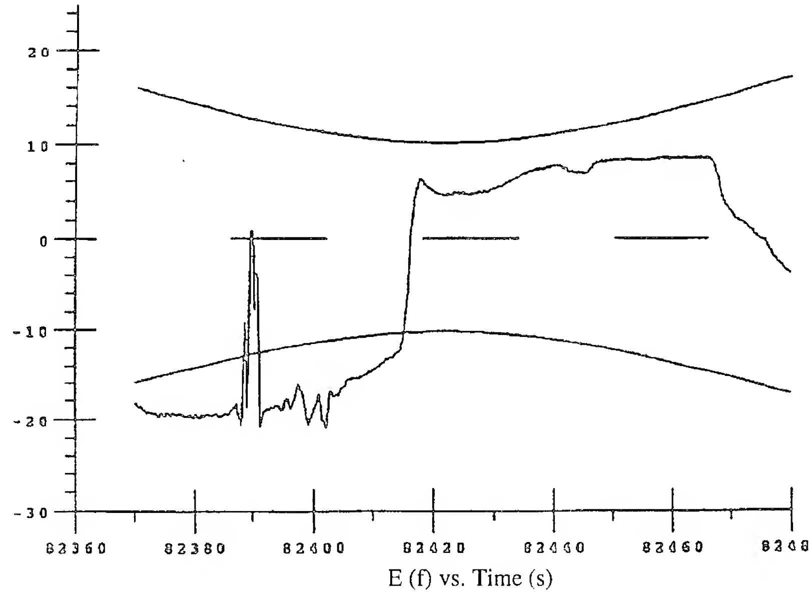
Figure 4.2.4-1b.

Position Segment 4

Solutions: The smoother errors (TANS+INS vs. cine-based reference) are shown in Figure 4.2.4-2. Similar to segment two, segment four shows ATDOP processing across average data gaps and the advantage gained using differential TANS aided with INS data. The data gap starts at time 82414.0 and lasts for 14 seconds. Note that even though there are maneuvers during the dropout, the errors remain bounded by 30 feet or less. ATDOP filter predicted uncertainty behavior for the TANS/INS results are similar to the results shown for the first two segments: TANS/INS TDOP solution uncertainties grow during dropouts and return to reduced values upon

resumption of TANS solution. Similarly, the smoothed uncertainties (and similarly the errors) are reduced during the dropout.

Residuals: TANS/INS residual behavior is similar to results shown for the first two segments, and are approximately white and lie within their predicted one-sigma values, thus indicating reasonable filter tuning. TANS residual uncertainties reflect the inflated measurement model used to account for unmodelled bias errors. The residual plots exhibit rejection of TANS data in certain regions and just before the data dropout region. For this segment, the effective data gap is approximately 20 seconds long due to TANS data rejection.



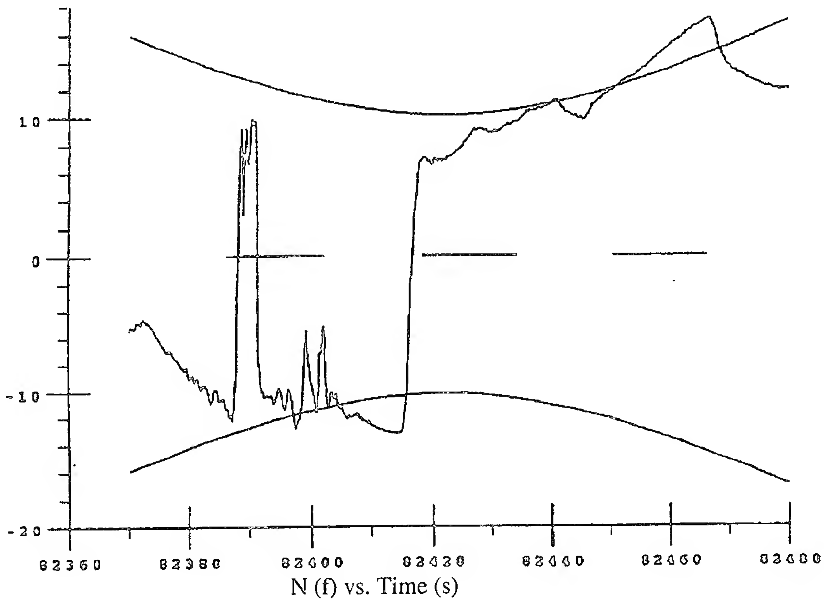
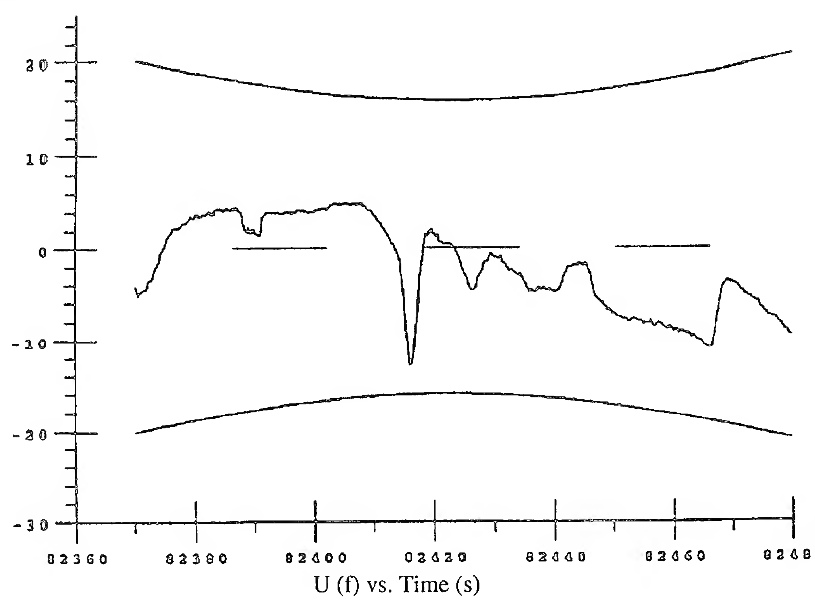


Figure 4.2.4-2.

ATDOP Smoothed TANS+INS vs Reference Segment 4



4.2.5 Segment Five

The flight profile for this segment is shown in Figures 4.2.5-1a (acceleration vs. time) and 4.2.5-1b (plan view and up vs. time) and exhibits maneuvers in both horizontal and vertical directions, due to a steep dive maneuver.

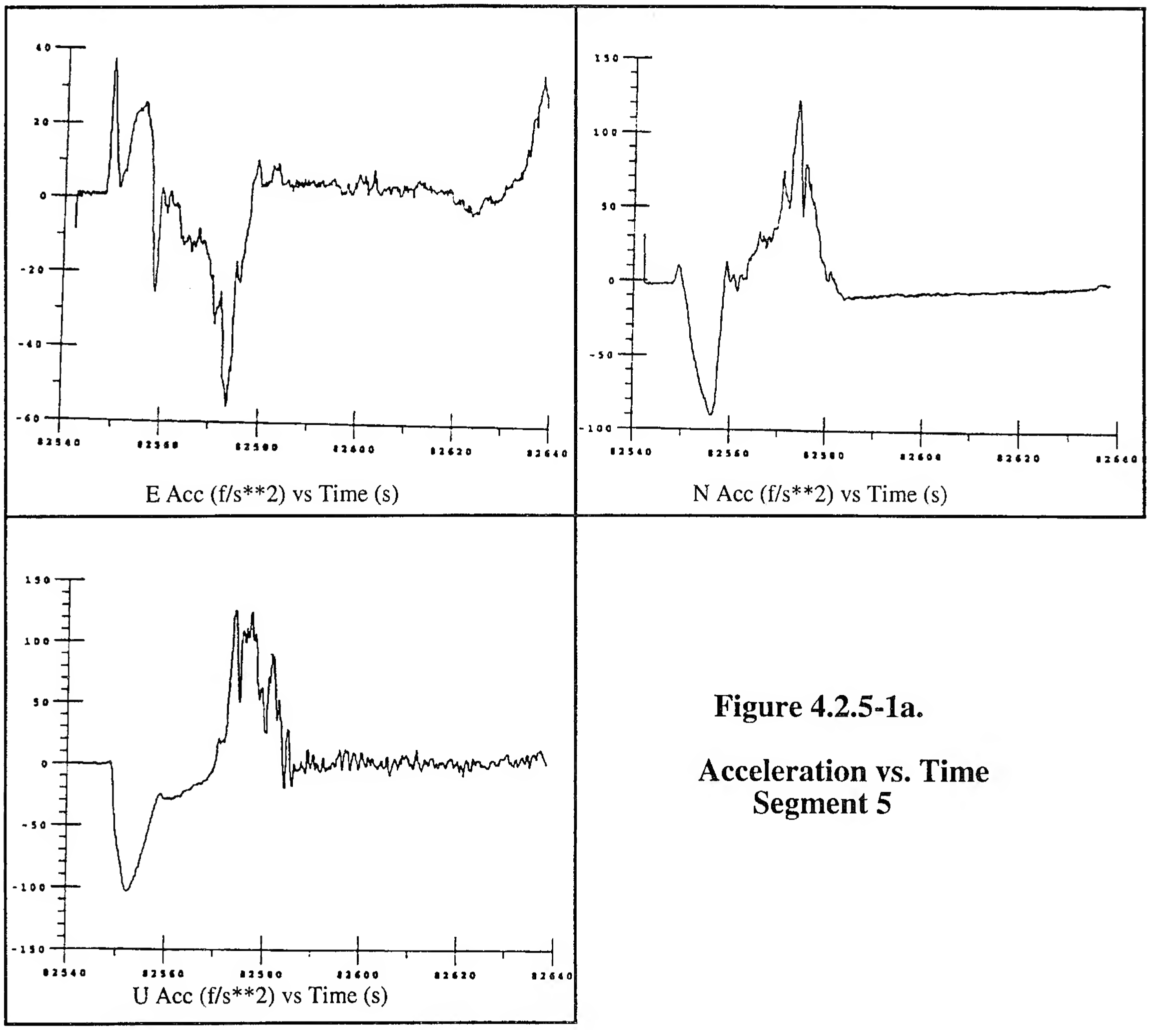


Figure 4.2.5-1a.

**Acceleration vs. Time
Segment 5**

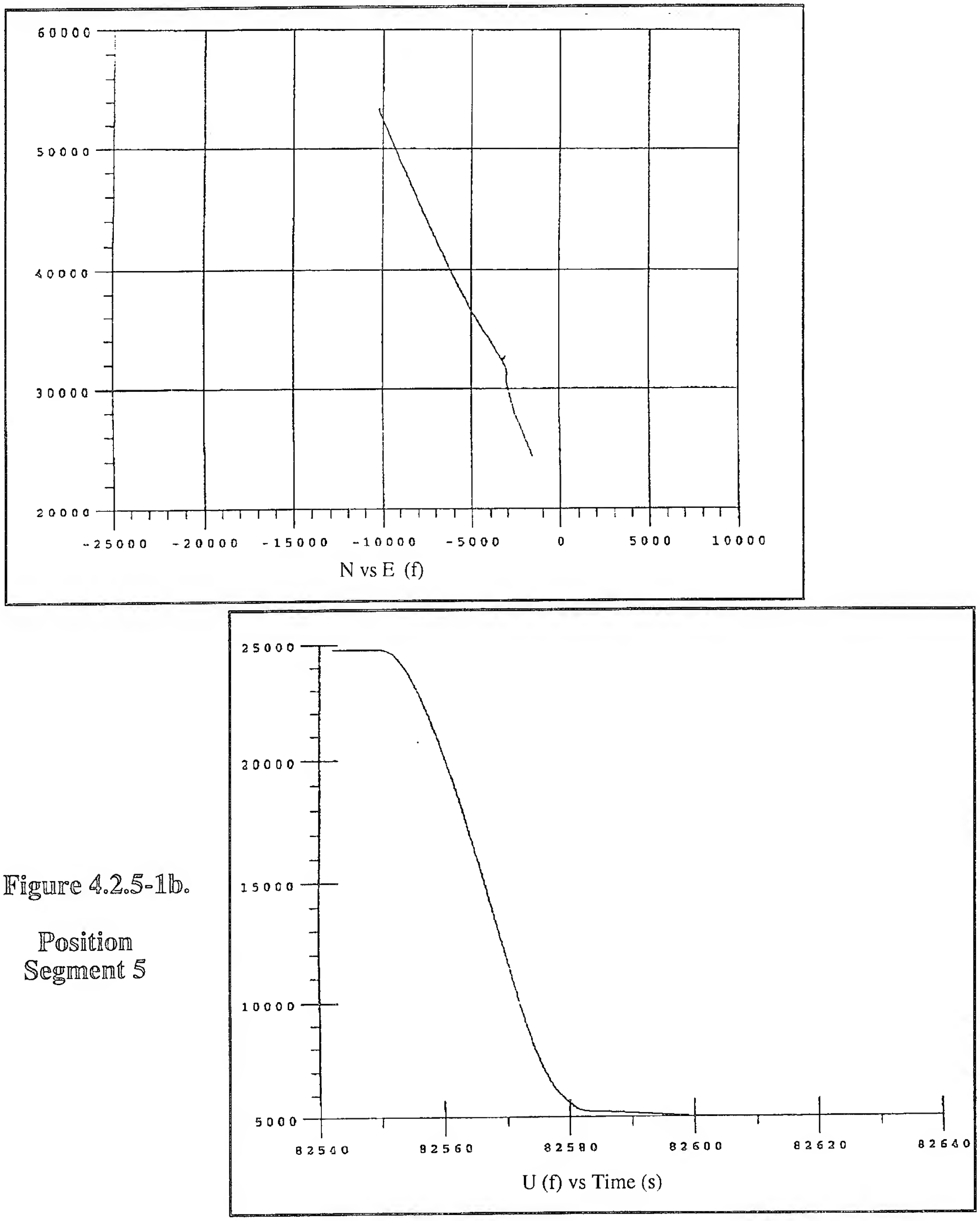


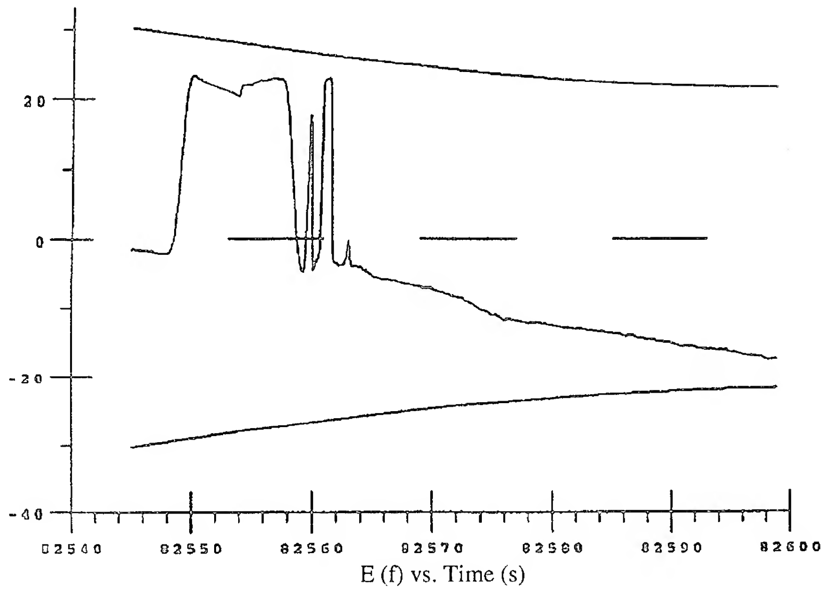
Figure 4.2.5-1b.

Position
Segment 5

Solutions: The actual errors (TANS+INS vs. cine-based reference) are shown in Figure 4.2.5-2. This segment shows ATDOP processing across gaps of above average length (> 20 seconds). The data gap starts at time 82555 and lasts for 30 seconds. During this period, ATDOP utilizes only inertial data. Note that even though there are maneuvers during the dropout, the errors remain bounded by 50 feet. ATDOP filter predicted uncertainty behavior for the this segment is similar to the results shown for the first two segments: TANS/INS TDOP solution uncertainties grow during dropouts, but very quickly return to reduced values upon resumption of TANS

solutions. Similarly, the smoothed uncertainties (and similarly the errors) are reduced during the dropout.

Residuals: Residual behavior for the TANS/INS results are similar to results shown for the first two segments, and are approximately white and lie within their predicted one-sigma values, thus indicating reasonable filter tuning. TANS residual uncertainties reflect the inflated measurement model used to account for unmodelled bias errors. The residual uncertainties grow during the data dropout period to reflect the absence of TANS measurements and drop back when TANS data resumes. Similar to the previous segments, ATDOP rejects a few data points just before the beginning of the data gap, thus making the effective data gap length approximately 35 seconds long.



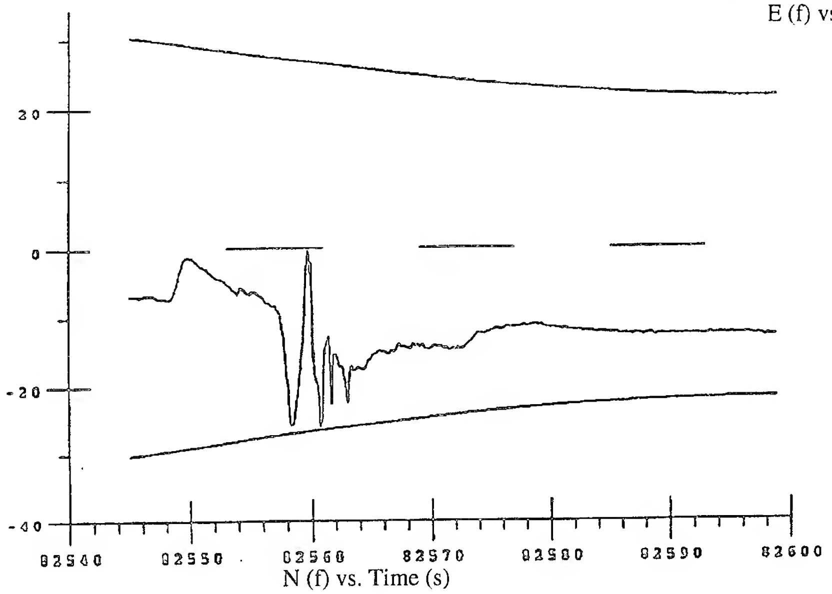
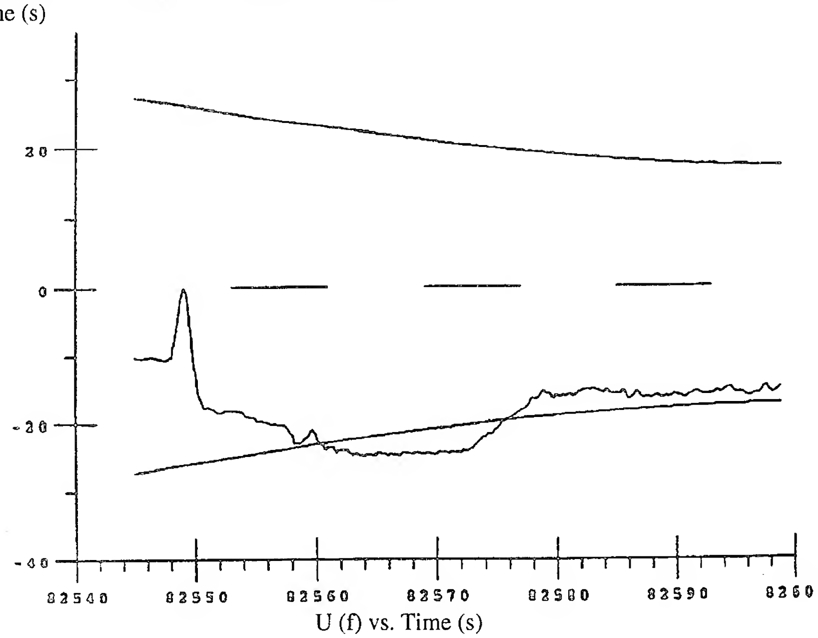


Figure 4.2.5-2.

ATDOP Smoothed TANS+INS vs Reference Segment 5



5.0 Conclusions

- ATDOP can be used to effectively smooth GPS receiver solution data for dropouts of 30 seconds and longer. GPS solution dropouts longer than 120 seconds can probably be handled as well. This is primarily a function of the quality of the INS data and the vehicle dynamics profile during the data dropout region.

- In the current experimental configuration, the TANS solution data display divergent behavior during maneuvers and prior to complete loss of track. ATDOP can be tuned to automatically reject these bad data. In addition, it is possible that the TANS receiver setup can be modified and set up in a different way to handle moderate to high dynamics situations more effectively. However, use of INS data and ATDOP processing can be used in combination with TANS data to provide trajectory information during receiver dropouts, thus reducing the criticality of such considerations. These results apply also to new GPS receiver technology (e.g., new JPO receivers, new receiver models from Trimble and other receiver manufacturers).

INSTRUMENTATION FOR THE DIGITIZED BATTLEFIELD

Patrick E. Clark
ACM Systems
9838 Old Placerville Road
Sacramento, CA 95827

Introduction - The Digitized Battlefield

Digitization of the battlefield represents the transition from analog to digital technologies. It will produce automated and integrated systems and operations from what use to be separate functions and activities. It involves the application of commercial and military information technologies, allowing vertical and horizontal sharing of data across wired and wireless networks. The digitization process organizes, analyzes, tailors and distributes information from sensor and supporting data collection assets that gather tremendous amounts of data which far exceed the needs and capacities of any individual.

The goal of digitization is to improve battlefield agility. This will be accomplished by *force synchronization, faster tempo* and *higher precision* through a combination of shared situation awareness, near real-time command and control, and improved sensor technology. Digitization of the battlefield exploits a variety of technologies, such as distributed database management, LPI communication, wireless LAN, global 3-D position location, image processing and data compression, expert systems and neural networks, and multi-spectral sensors.

Test and Evaluation Issues

The hardware and software development of the digitized battlefield is an evolving process, primarily due to the complexities of the system and subsystems, as will as the expected growth and change associated with information technology. Methods of test and evaluation must reflect these uncertainties by increasing equipment flexibility through the use of modular and reprogrammable techniques[1].

The Army plans to implement the digitized battlefield through a series of

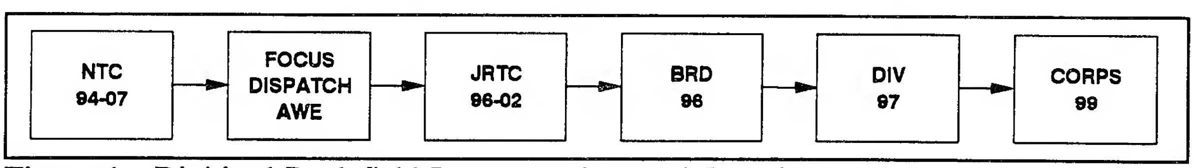


Figure 1. Digitized Battlefield Demonstrations and Experiments.

[1] Treating test equipment and methods as consumable has also been suggested but does not appear to be cost-effective due to high design costs and extensive development schedules.

demonstrations and experiments as shown in Figure 1. This began with Desert Hammer VI, which was conducted during April 10-23, 1994 (NTC Rotation 94-07) and demonstrated the potential of the digitized battlefield. Focus Dispatch AWE is scheduled to be conducted during the Summer of 1995 and will primarily involve heavy armor (M1/M2). Warrior Focus will evaluate dismounted troops outfitted with voice/data radios. JRTC will combine dismounted troops, heavy armor and the prototype C2V. Brigade 96 (BDE 96) will demonstrate the improved interfaces between systems by allowing the horizontal exchange of information. Battalion to brigade interfaces for MCS and CSSCS will also be implemented. For Division 97 (DIV 97), a common protocol (MIL-STD-188-220) and a common message set based on the variable message format will be added [2].

Test and evaluation must address many technical and operational issues concerning the digitized battlefield. Five specific digitization areas are discussed below.

Overall Mission Improvement. The intent of digitization is to improve warfighting capability. Although this result should naturally flow from the orderly integration of digital and network technologies, this has yet to be demonstrated in a large scale field exercise. Of primary concern is that digitization will create more technical and operational complexities for the battlefield decision makers that will impede rather than help the overall mission. These effects stem from the mental stress on the soldiers caused by information overload.

Multiple Interfaces and Protocols. Within the combined arms scenario, participating combat systems use different Combat Net Radio (CNR) protocols, data formats and data element

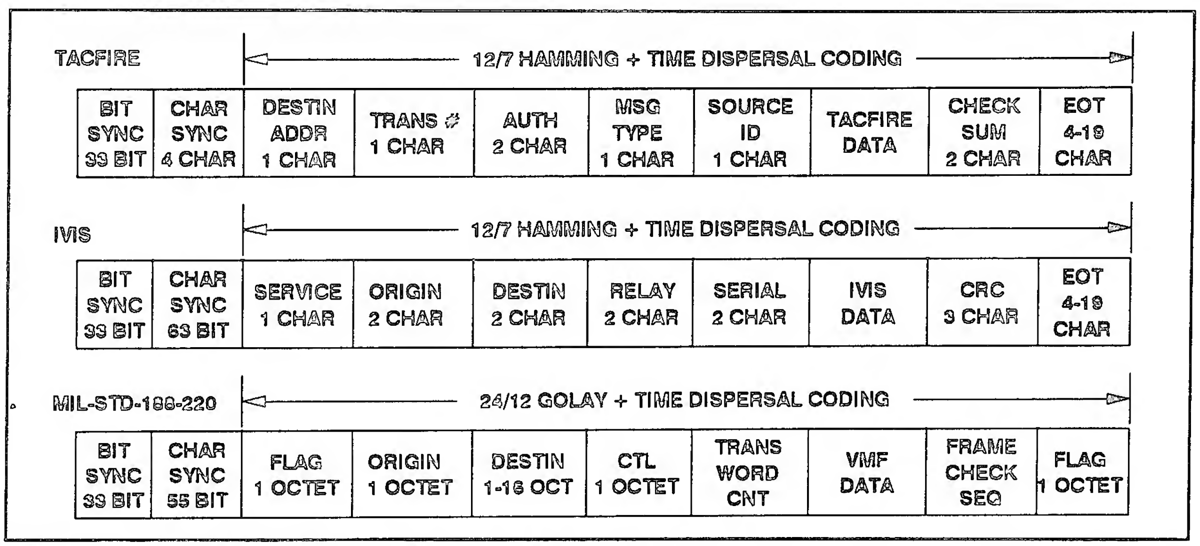


Figure 2. Combat Net Radio Protocol Comparison.

structures. Figure 2 compares three message formats, although many more (B2C2, ATCCS, INTEL, ARTY, etc.) can be added. However, in order to achieve the objectives of digitization, information must be acquired, tailored, exchanged, and employed by a variety of users, necessitating frequent translations from one format to another. Potential problems include loss of information or precision, and increased communication delay.

Throughput Capacity and Message Latency. Previous Army studies showed the need for multiple communications systems to meet the communication (voice) and digitization requirements. These systems - SINCGARS, EPLRS and MSE - are expected to supply the bandwidth and support the data rates necessary for brigade area users. Data exchanges between any of these 1200 users should not exceed several seconds [1]. However, since the implementation of the digitized battlefield is an evolving process and new requirements are expected to appear, data capacity may be exceeded, resulting in unacceptable latencies.

Electronic Countermeasure (ECM) Vulnerability[2]. The enemy has access to a large arsenal of ECM weapons, including high power standoff radiators, fast frequency followers, artillery-launched jammers, etc. Modern communication systems incorporate a variety of anti-jam techniques, such as spread spectrum, power management, steerable beam/null antennas, and error detection and correction codes. Separately or in combination, these techniques can greatly reduce the ability of a hostile transmitter to disrupt a communication or data link. However, each technique relies on assumptions regarding the jammer's location, processing speed, modulation type or transmitter capacity (see [4] and [5] for a discussion of jamming effects on frequency hopping and direct sequence spread spectrum radios). As experienced in commercial computer networks, hackers can insert viruses, worms, and other spoofing devices, without detection. Evidence of network security breaches of this type often occur only when these devices are activated. Regardless of the method used to disrupt the battlefield network, performance degradation should be expected.[3]

Training and Soldier Proficiency. The digitized battlefield requires that soldiers be proficient in digital systems as well as warfighting fundamentals. As the digitized battlefield evolves, sustainment training must include a means for updating the soldier's knowledge and understanding of the network's operation and maintenance. Methods of assessing proficiency must also be provided so that equipment and training inadequacies can be recognized and corrected in a timely manner.

Digital Data Collection Techniques

Figure 3 shows a portion of the digital network installed within a ground-mobile platform[4]. SINCGARS is a frequency hopping VHF radio that is capable of maintaining voice and digital data links. It represents the external user connection to other network participants. SINCGARS protocol consists of collision detection capability with receiver acknowledge/not acknowledge (ACK/NAK) responses to messages. If no response is received, the transmitter performs multiple retries until time-out occurs. When installed on

---

[2] Besides possible ECM vulnerabilities, network outages can also result from a number of other causes. Equipment that is severely stressed due to the environmental effects of the battlefield may fail. Physical destruction of assets is possible. Mobile operation over adverse terrain also contributes to momentary link disruption.

[3] Vulnerability to ECM is the subject of a future paper.

[4] This is a portion of the Intervehicular Information System (IVIS) installed on a tank or BFV.

the M1A2, two radios are provided in order to maintain two simultaneous links. The Radio Interface Unit (RIU) provides the interface between SINCGARS and the MIL-STD-1553B data bus. It maintains the Network Radio Protocol, reformats between SINCGARS and the internal net, filters messages according to platform ID number, and performs error detection and correction. The Commanders Integrated Display (CID) provides the man-machine interface and handles the reception and generation of all reports and overlays. The Turret Electronics Unit (TEU) maintains the platforms database, provides standard formats for reports and overlays, and functions as the MIL-STD-1553B bus controller.

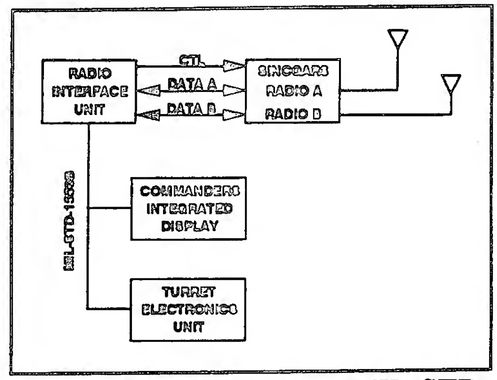


Figure 3. SINCGARS to MIL-STD-1553B Interface.

Two possible locations for digital data collection are shown in Figure 4. A previous paper [6] provides a detailed description of the platform, MIL-STD-1553B data bus, and the Multi-Purpose Data Collector (MDC). This paper will describe the design, installation and performance of the Field Data Collector (FDC) and its successor, the Improved Field Data Collector (IFDC).

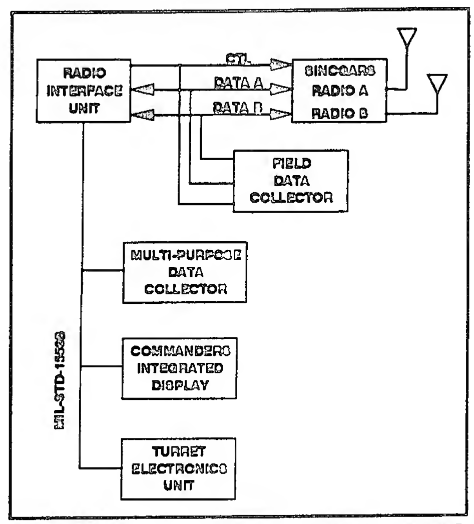


Figure 4. MDC and FDC Interfaces.

The Field Data Collector

The purpose of the MDC was to monitor the internal data bus. However, several shortcomings exist with this configuration. First, the RIU ignores all messages except those addressed to the platform's ID, so that a missed message will not be recorded. Furthermore, since the platform's ID is unique, the message will not be recorded anywhere in the network (except at the transmitting node if it is assumed to be operating properly). The data collection system cannot determine why the message was missed - the platform may have ignored the message because it was overloaded with voice or other digital traffic. Second, the RIU deletes properly addressed messages which have been corrupted, i.e. they contain a bad parity bit or CRC string. During tactical situations these messages are obviously ignored, but for test and evaluation purposes, they contain useful information regarding propagation characteristics and error handling protocols.

The FDC was developed to overcome these problems. By placing the FDC at the location shown in Figure 4 and recording data whenever a valid clock transition is detected, the FDC functions as a network data collector. This is because all received network transmissions with the appropriate hopping code are de-hopped and demodulated by SINCGARS and sent to the RIU regardless of user ID number. The functional requirements for the FDC include:

1. Record NRP data from two independent channels of tactical radio data[5]
2. Separate data streams into messages
3. Time tag messages
4. Merge data and store in a removable medium
5. Measure and record platform position
6. Operate as a transparent bus monitor
7. Provide reliable performance in a rugged exposed field environment
8. Minimize size and weight.

The FDC records NRP data from two identical NRP channels (A and B) and

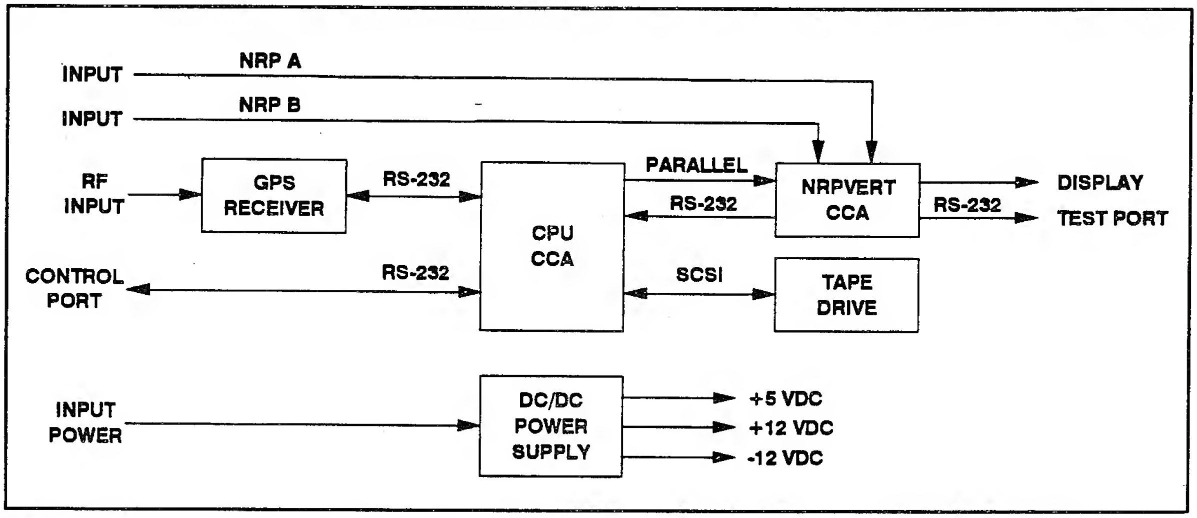


Figure 5. Field Data Collector Block Diagram.

processes these inputs even when both are simultaneously active at 9600 baud[6]. Figure 5 illustrates the internal design of the FDC. The design incorporates four off-the-shelf (OTS) assemblies (six-channel C/A code GPS receiver, CPU CCA, Tape Drive and DC/DC Power Supply) and a single custom circuit card (NRPVERT CCA). The NRPVERT CCA converts and merges the NRP data to a single RS-232C serial data stream. The serial data is input to the single board computer which runs the embedded software. Time tagging and position data are produced by the GPS unit which is connected to the CPU though a second serial port. Data is collected and sent to buffers which are periodically written to tape for storage. An additional external port can be connected to a remote terminal for status and configuration purposes.

---

[5] The FDC collects data transmitted between RIUs and SINCGARS radios A&B when installed on M1A2 tanks and/or between Lightweight Computer Units (LCUs) with Tactical Communications Interface Modules (TCIMs) and SINCGARS Radios A&B when installed on other vehicles.

[6] The FDC also supports single channel operation up to 16000 baud.

Data Stream Processing. SINCGARS channels A and B are used for both analog voice and data communications. The analog portion is ignored by the FDC. Each channel's digital clock is monitored and, upon sensing a transition, searches the bit stream for alternating ones and zeros that are characteristic of the bit synchronization pattern at the start of message. Data direction (receive or transmit) is determined and the FDC inserts a Start of Message byte followed by the data. Each recorded byte consists of six data bits, a status bit and radio A/B designation bit. The latter two bits are created by the FDC. After receiving all of the data bits, the FDC adds an End of Message byte, Count byte[7], and remaining data bits. If a channel is producing data at 9600 baud, the additional overhead (2 bits per 6 data bits) creates an effective 12800 baud data stream. The FDC adds appropriate start and stop bits for RS-232C compatibility, resulting in a formatted bit stream of 19200 baud. Data from both SINCGARS channels are merged into a single serial stream operating at 38400 baud. The FDC's internal tape drive supports a recommended tape length of 950 feet, which is equivalent to 1 Gigabyte of storage capacity.

NTC 94-07 Performance. The FDC was developed to support Desert Hammer VI (NTC Rotation 94-07). This exercise consisted of 18 M1A2s, 6 M2A2s (with IVIS), 1 M2A3 (with B2C2), 66 LCUs (with B2C2) and 3 digitized dismounted soldiers equipped with prototype hardware. Through a combination of instrumentation and observation, several insights were gained [3]:

1. The use of different protocols (TACFIRE, B2C2, IVIS, etc.) created major translation problems.
2. Contingency plans are necessary for re-establishing lost links due to the enemy fire and link deterioration.
3. Digital connectivity must be established prior to the start of the exercise since many link-ups during operations were not successful.
4. Key non-vehicular positions (dismounted troops and fixed installations) were not added to the digital displays which impacted calls for fire.
5. Some participants did not utilize their digital assets to their full potential.
6. Additional troop training is required and more automation of repetitive tasks must be implemented.

The Improved Field Data Collector

As a result of the experience gained during NTC 94-07 and the requirement for additional demonstrations and experiments, the FDC is being upgraded by expanding its flexibility and data processing capabilities. The new device, called the Improved Field Data Collector (IFDC), supports SINCGARS and EPLRS, performs more statistical preprocessing, contains a telemetry port and a combination telemetry/download port, provides for remote control, has differential GPS capability for greater positional accuracy, and improves time tagging accuracy to 1 millisecond. Figure 6 illustrates the assemblies that comprise the IFDC.

---

[7] The Count byte indicates the number of remaining bits under six in the last data word.

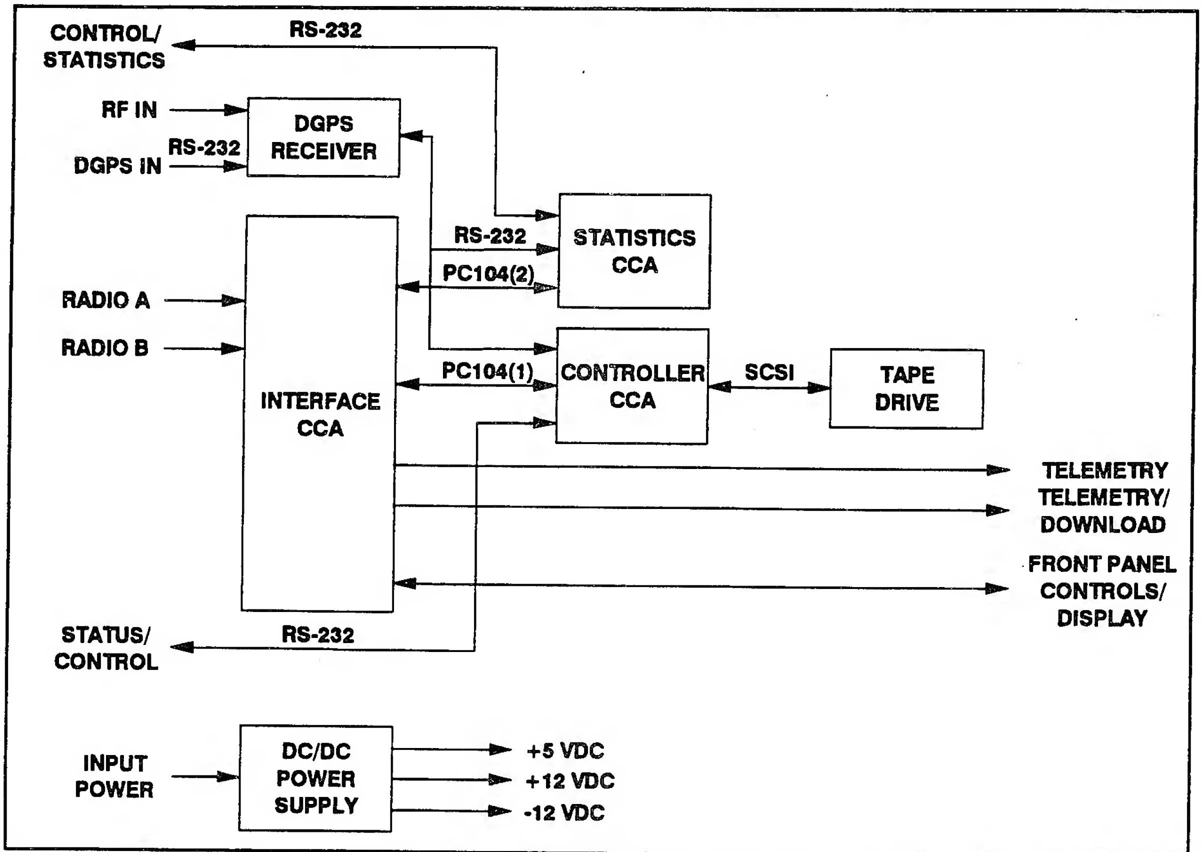


Figure 6. The Improved Field Data Collector Block Diagram.

Multiple Radio Capability. The IFDC can be reconfigured, through software control, to accommodate all combinations of SINCGARS and EPLRS. The IFDC will read, process and record SINCGARS NRP and EPLRS protocols, and contains the hardware needed to receive and record MIL-STD-188-220 signals for Applique as well as RS-423 and RS-422 balanced and unbalanced data communications inputs. Data rates have been increased to a maximum of 32000 baud and signals can be modulated according to either NRZ or CDP (conditioned di-phase) formats.

Statistical Preprocessing. The IFDC will perform a limited amount of message processing that, in conjunction with a new remote control and status reporting capability, will allow near real-time monitoring of the exercise. Message processing will consist of examining and extracting the message fields for destination, origin, message type and relay destination. The IFDC will add status (radio A or B, transmit or receive mode) and store the resulting 10 byte message in a buffer. Data will accumulate in this buffer for 10 seconds and, when requested, its contents will be sent to the IFDC's Control/Statistics port. The IFDC will create a header that contains a time tag, IFDC health and platform position (if requested).

Telemetry and Downloading. The IFDC will provide "as it happens" outputs of unprocessed data. Either port can be configured as an RS-422 or RS-423 interface and supplied with I/O hardware capable of supporting maximum data rates of 2 Mbps and 19.2 kbps, respectively. The primary use of these ports is to transmit the contents of the data log to a central location

for overall exercise review. Remote data log extraction eliminates the need for technicians to access the platform and physically remove its tape. Besides the inconvenience and difficulties involving with traveling to these sites, tape removal requires opening the protective equipment enclosure which exposes the internal electronic and mechanical assemblies to excessive dust and moisture.

Remote Control. Remote control of the IFDC anticipates the integration of a platform-mounted real-time casualty assessment (RTCA) system[8] that contains the hardware necessary for wireless operation. When downloading is required, the IFDC system will receive control commands over the RTCA link to stop recording radio data, rewind the tape and prepare to download the data.

Differential GPS. The IFDC contains an upgraded OTS GPS receiver that supports both stand alone and differential modes of operation. The primary purpose of this upgraded capability is to improve the position location accuracy by an order of magnitude even when Selective Availability is active[9].

Improved Timing Accuracy. Time tag accuracy has been improved by two orders of magnitude (100 vs 1 millisecond for the FDC and IFDC, respectively) in order to allow investigation of network timing conflicts.

The IFDC has been designed to assist in the evaluation of the Digitized Battlefield beginning with JRTC 96-02 and continuing through DIV 97 (and CORPS 99). It contains sufficient flexibility through software reconfiguration to meet the anticipated needs of new radios, protocols, message formats and data elements. Considerable on-board non-volatile storage allows unattended operation for at least 14 days at typical data collection rates. However, provisions are available for remote extraction of all or part of the data log.

Conclusion

The Digitized Battlefield will leverage currently available advanced information technology. The test and evaluation community will help direct these systems towards optimization of mission effectiveness, message throughput and soldier training. Development and integration of the instrumentation necessary to measure the necessary parameters has began and is currently in its third generation. Like the implementation of the Digitized Battlefield, test and evaluation instrumentation is also evolutionary and will continue to change and adapt as the system matures.

---

[8] The Mobile Automated Instrumentation Suite (MAIS) is an example.

[9] Selective Availability (SA) allows the DOD to intentionally degrade GPS C/A code accuracy.

Bibliography

1. McChesney, J., "Comments on Communications for the Digitized Battlefield", *Proceedings of the 1994 Tactical Communications Conference*, May 1994, v. 1, pp. 101-107.

2. Liggett, C.K., "Digitizing the Battlefield", *Proceedings of the 1994 Tactical Communications Conference*, May 1994, v. 1, pp. 115-118.

3. Shevely, R., "Battlefield Digitization", *AFCEA Panel Sessions, IEEE Military Communications Conference*, October 1994, pp. III:46-III:72.

4. Munday, P.J., et al, "Jaguar-V Frequency-Hopping Radio System", *IEE Proc.*, June 1982, v. 129, n. 3, pp. 213-222.

5. Torrieri, D.J., "Frequency Hopping in a Jamming Environment", *Technical Report*, CM/CCM-78-2, December 1978.

6. Clark, P.E., "Digitization of the Battlefield: Test Requirements, Methods, Equipment and Analysis", *Proc. TestSym VII*, Laurel, MD, February 1994.

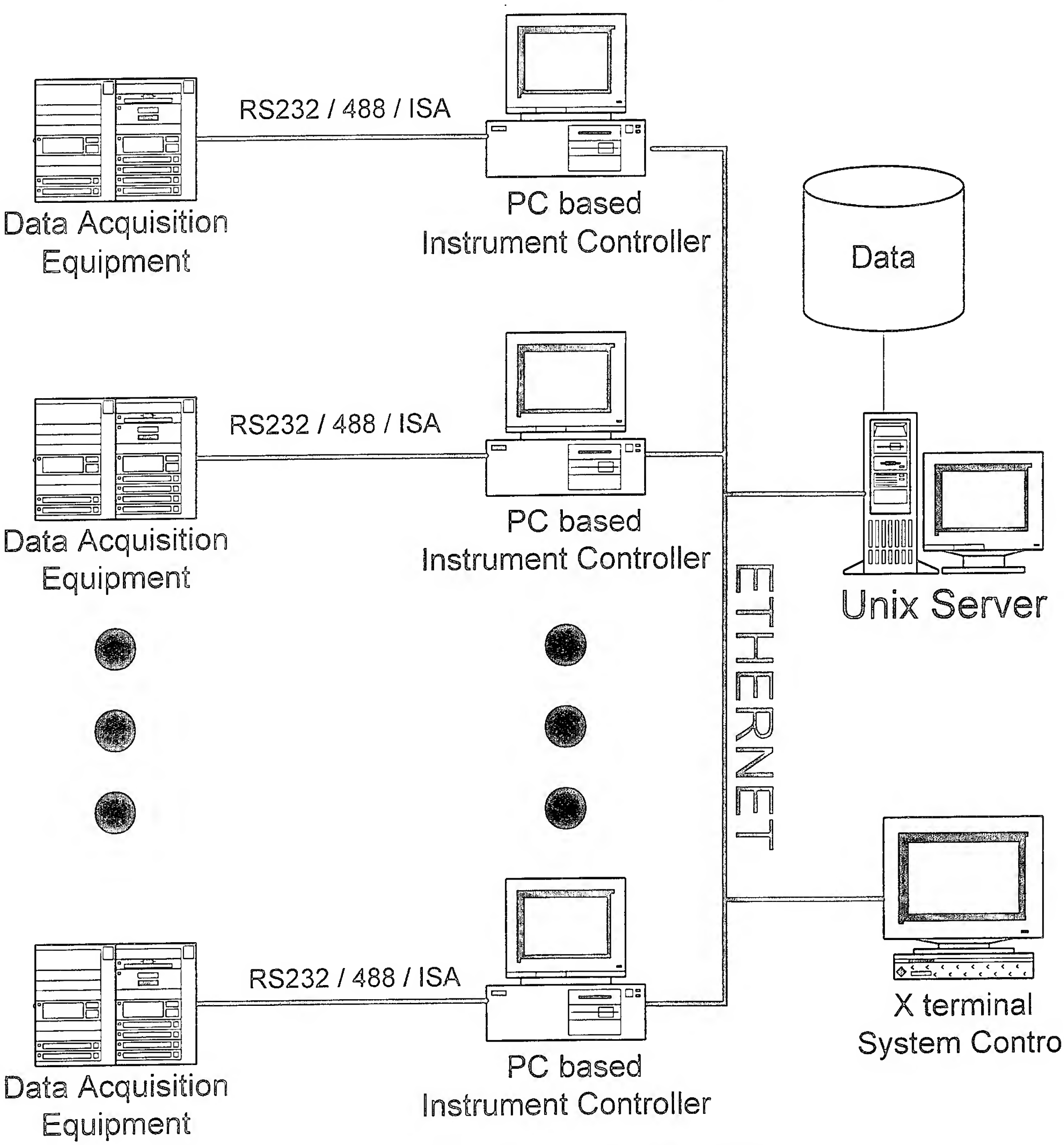
A Low Cost Instrumentation Data Collection System over Ethernet

Paul Storey
SM-ALC/TIEBD
4235 Forcum Ave Ste. 1
McClellan AFB, CA 95652-1504
av 633-4762, 916-643-4762
storey@sm-eise.af.mil

ABSTRACT:

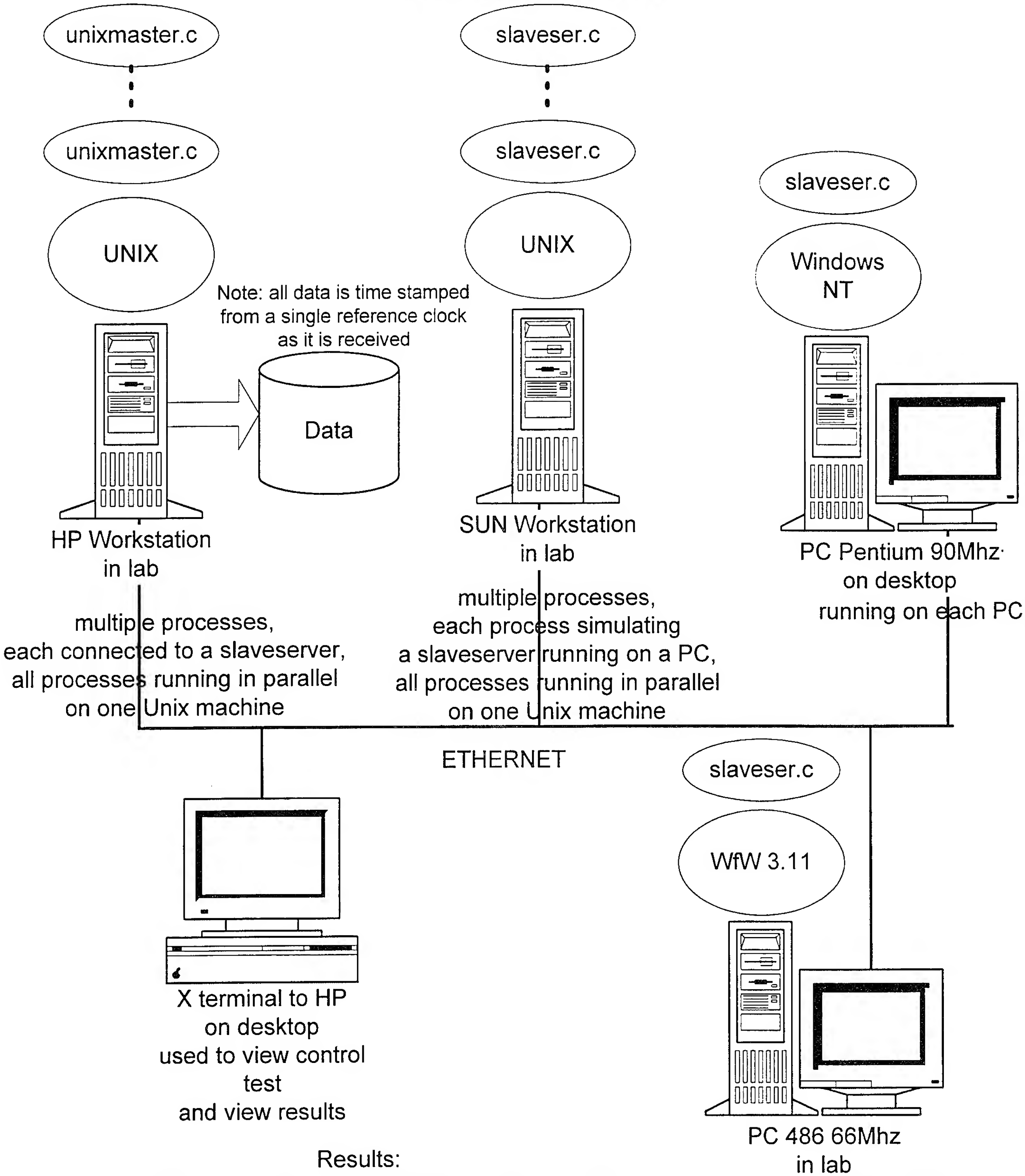
The Software Technology Integration Center in the Technology Integration Division at Sacramento Air Logistics Center is involved in development of a data acquisition and control system. This system is being developed for a dual use project to be used as the communications backbone for test equipment distributed over a network. The data acquisition system will use standard commercial off-the-shelf cards over Ethernet link. A typical system will consist of 40 PC or X86 embedded controllers feeding data to a single UNIX based data collection master over 10BaseT thinnet at 10 samples per second. The communications protocol is TCP/IP. We are able to transfer a 1024 byte packet from a single PC to a UNIX machine in a millisecond, which includes all overhead of operating systems on both ends. The architecture will scale well because collisions of CSMA are avoided due to a master slave relationship which provides the determinism needed for real-time systems. The network and transport layers of the OSI network model are complete. Application layers such as control state transition diagrams and mechanisms of time clock synchronization and are currently under development. The system should be extremely applicable to many government and commercial uses due to its $100 per node cost for Ethernet hardware and under 10 pages of C source code, which will be available license free.

Physical Architecture of Ethernet Based Data Acquisition System



Requirements: 50 PCs
Transfer 1K byte to Server
10 times / sec

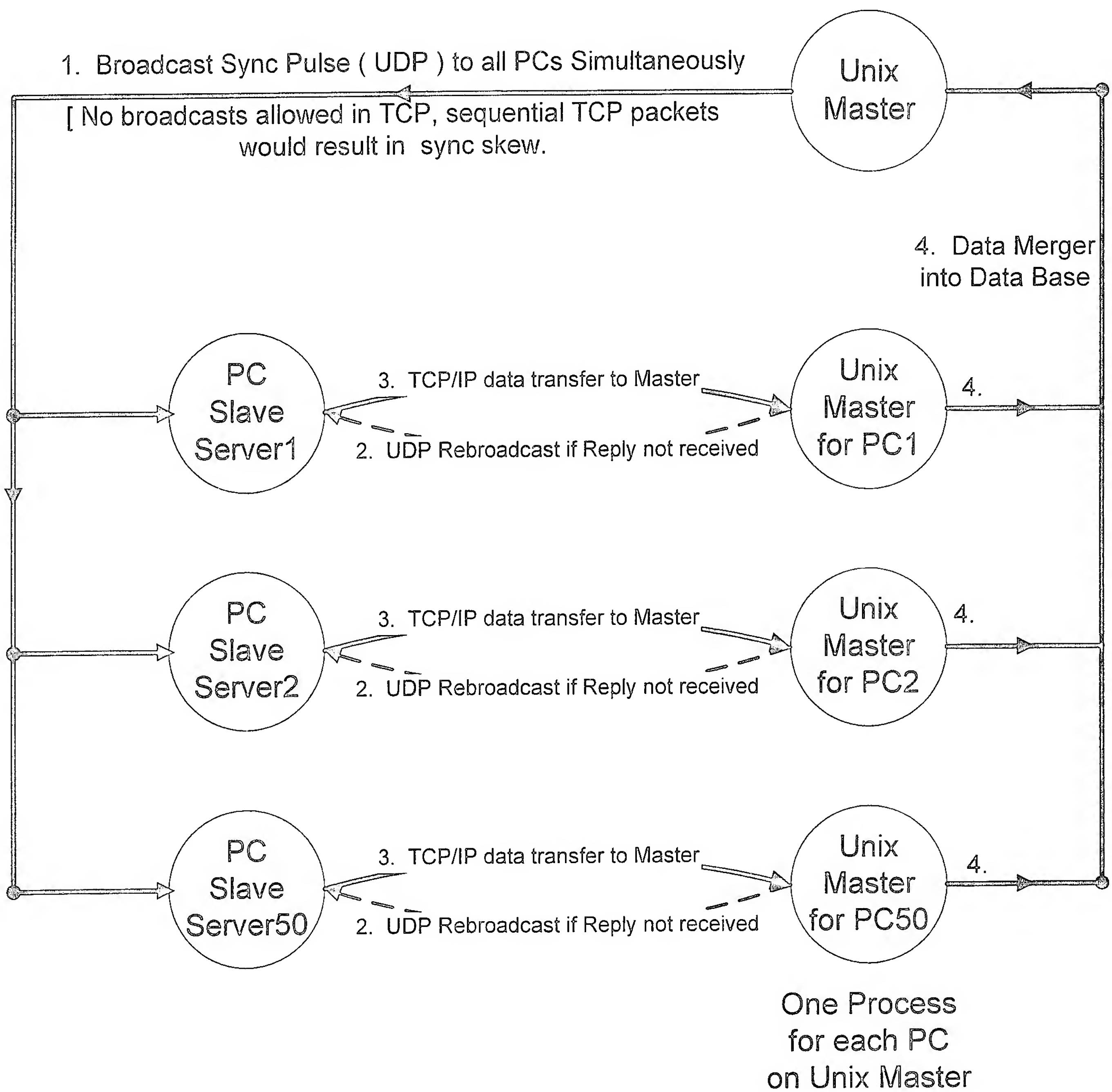
Instrumentation Data Collection System over Ethernet



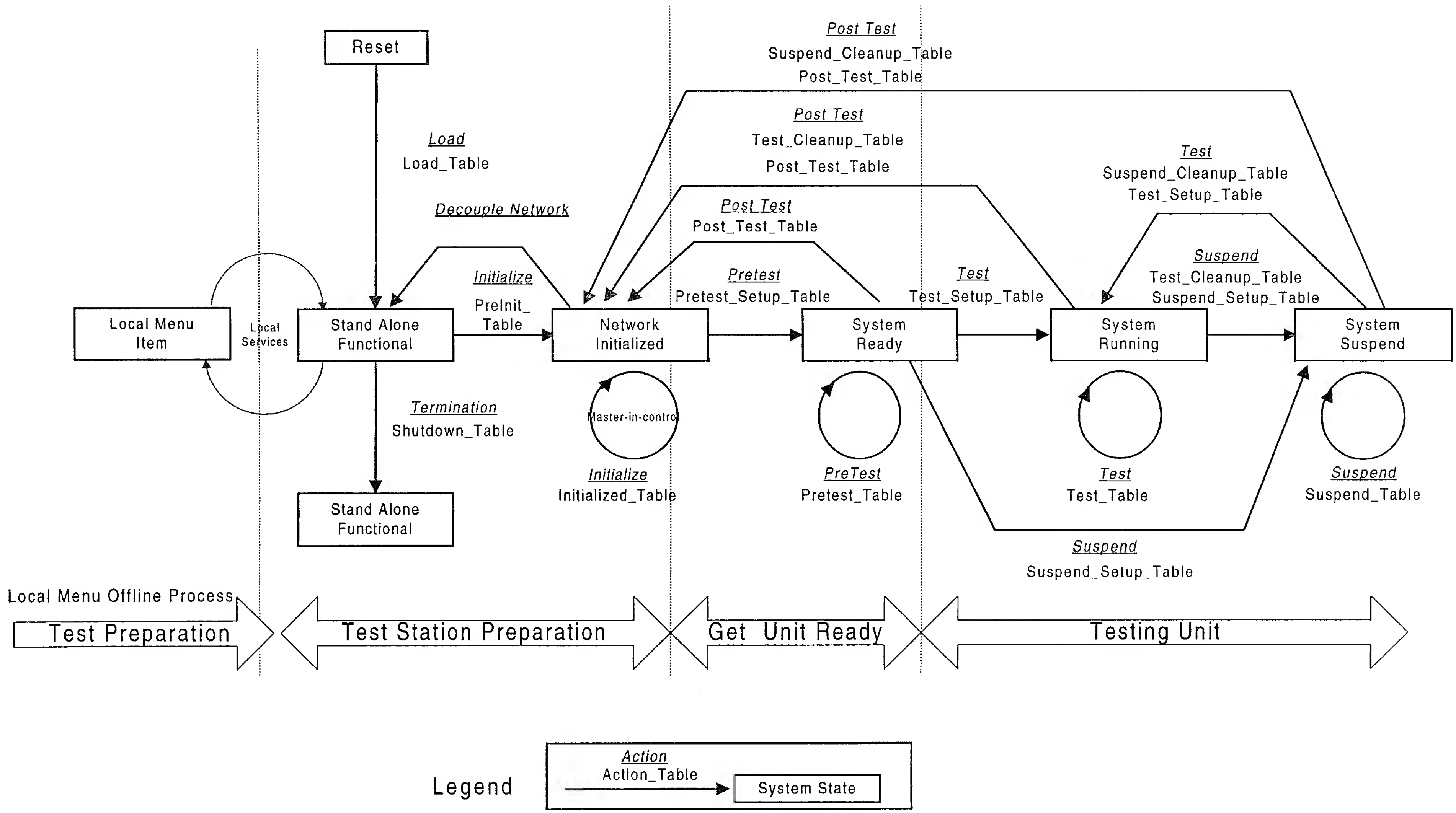
Results:
one 1024 byte packet transferred in 1 msec on PC
one 1024 byte packet transferred in .9 msec on UNIX machine
transferred to the UNIX master

SM-ALC/TIEFA
GW2\C:\AIGER\
TCPTEST6.VSD
P. Storey
6 June 95

Control and Time Synchronization: Master - Slave is Deterministic. No CSMA Collisions



Instrumentation System Test Functional Flow with Action Tables



569

Action Tables

Load_Table
o Initialize programs on each component
o Setup network & addresses
o Broadcast capabilities & configuration
o Load operating system
o Initialize all local mode capabilities
o Run "Local" mode menu actions

Initialized_Table
o Menu Selections
o Master/Slave keep alive

Pretest_Table
o Menu Program
o Test Run
o Debug

Test_Table
o Execute selected tests
o Coastdown
o Test profiles
o Sync Master
o Scheduler
o I/O Routines
o Calculations
o Data Recording
o Time Tagging

Suspend_Setup-Table
o Optional Test dependent actions

Suspend_Cleanup_Table
o Optional Test dependent actions

Shutdown_Table
o Close all files
o Power off all hardware

PreInit_Table
o Initialize all remote mode capabilities
o Initialize Menu Program
o Remote Diagnostics
o System Calibration
o Initialize Master/Slave relationships

Pretest_Setup_Table
o Prepare Vehicle info
o Mount Car on Dyno
o Purge lines

Test_Setup_Table
o Initialize distributed test sequence/memory
o Initialize Site Test parameters
o Load Vehicle data
o Mechanical Setup
o Start car
o Short duration things

Test_Cleanup-Table
o Optional Test dependent actions

Suspend_Table
o Test dependent actions

Post_Test_Table
o Save collected data
o Secure/shutdown car
o Safely secure all moving parts

Decouple_Table
o Close all sockets
o Kill all processes

1 ❑ **A Low Cost Instrumentation Data Collection System Over Ethernet**

TWFGCon 95
15 June 95

Paul Storey
SM-ALC / TIEFA
McClellan AFB, CA
916-643-4762
storey@sm-eise.af.mil

2 ❑ **500 Sample per Second Data Acquisition Evolution:**

- 1960 Analog Discrete Components / Multiplexed onto analog tape
- 1970 Chips A/D, D/A
- 1980 VME cards,
- 1985 VXI Instruments on a card
- 1990 PC based Controller and GUI

3 ❑ **Present Requirements**

- up to 50 PCs gathering data
- data collected in central DB
- 10 samples per sec
- low cost
- good user interface

4 ❑ **Approaches:**

- fifty RS-232 too many hardwired cables
- IEEE 488 length limitations
- Scramnet $5k per machine
- Ethernet SW overhead and collisions

5 ❑ **Ethernet Cost:**

- Network cards INEXPENSIVE
- Network cabling CHEAP
- Protocol stack FREE
- Custom SW ZERO

6 ❑ **Ethernet Performance**

- 10 M bits / sec = 1.2 M bytes / sec
- CSMA efficiency decreases as traffic increases
- non deterministic
- not for real time

7 ❑ **SW Development**

- Evolutionary Development
  - » Proof of Concept
  - » Try it and see
- c language
- SEI level 2+ process and coding practices
  - – Daily planning
  - – Weekly code walkthroughs
  - – Quality Control

8 Performance Measurements

- one packet (1024 bytes) per millisecond
- 1M bytes per sec
- sustained 80% maximum Enet BW

9 Mimick 1553 Protocol

- Master - Slave [command - response bus]
- Slaves wait until master issues command
- deterministic
- 1 command for 1 response = 100% overhead

10 Time Synchronization:

- Problem:
  - PC Clock drift unacceptable
- Solution:
  - UDP broadcast sync pulse to all PCs simultaneously
  - PCs collect data, when completed, respond with TCP/IP packet to master
  - No time skew in start of cycle
  - fallback to polling if too many collisions

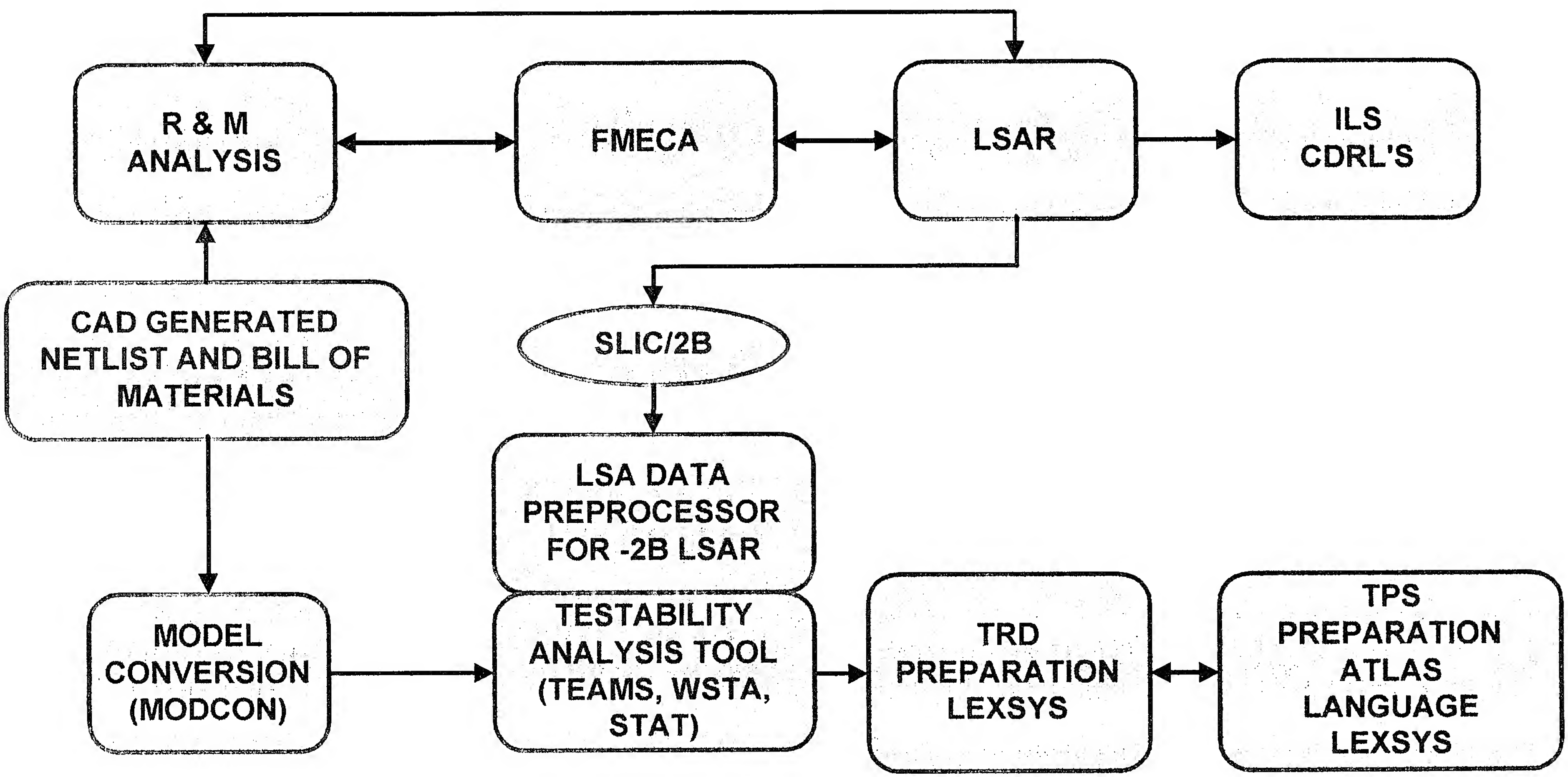
11 End Result:

- We beat Technoflation
- well defined goals
- project 60% complete
- best flattery = emulation
  - used by Dr. Sha, SEI

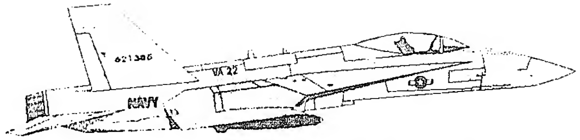
12 Summary:

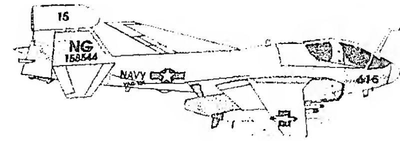
- Low cost
- Deterministic
- Readily available

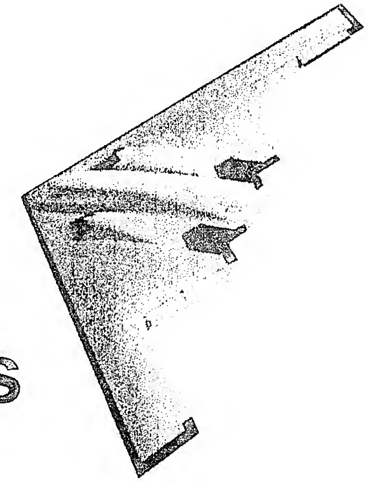
A Totally Integrated and Linked Logistics Capability



AIL SYSTEMS INC.
TECHNICAL SERVICES OPERATIONS

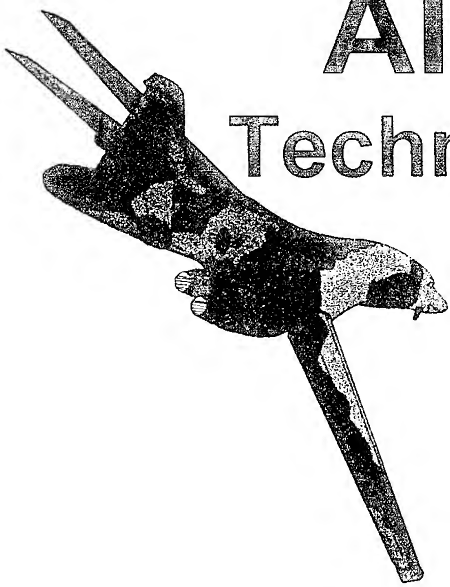




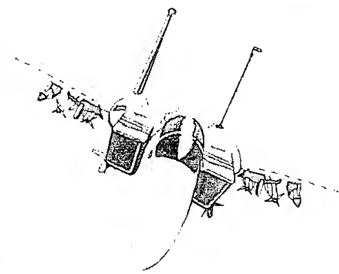


AIL Systems, Inc.

Technical Services Operations



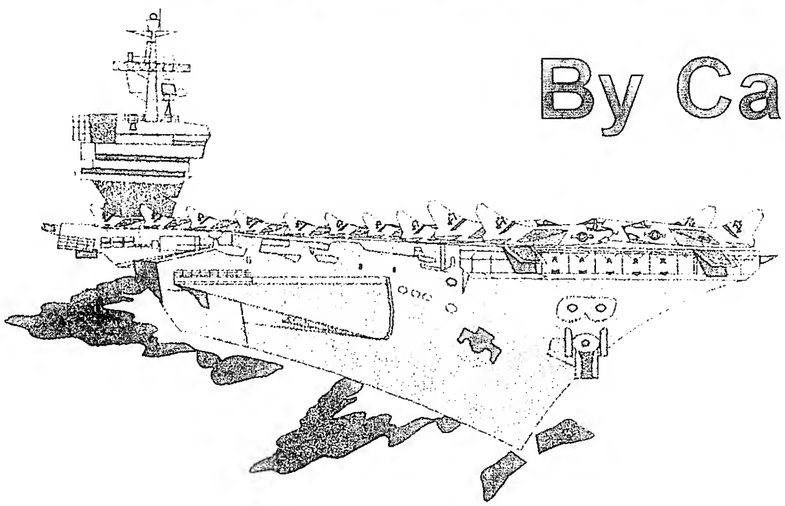
TFWGCON '95

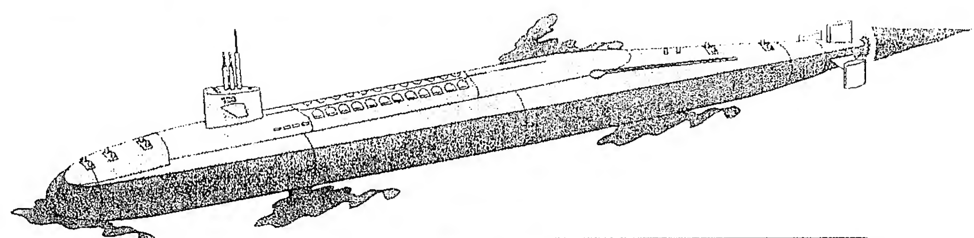


Lean Logistics

Low Cost "Pseudo Organic Capability"

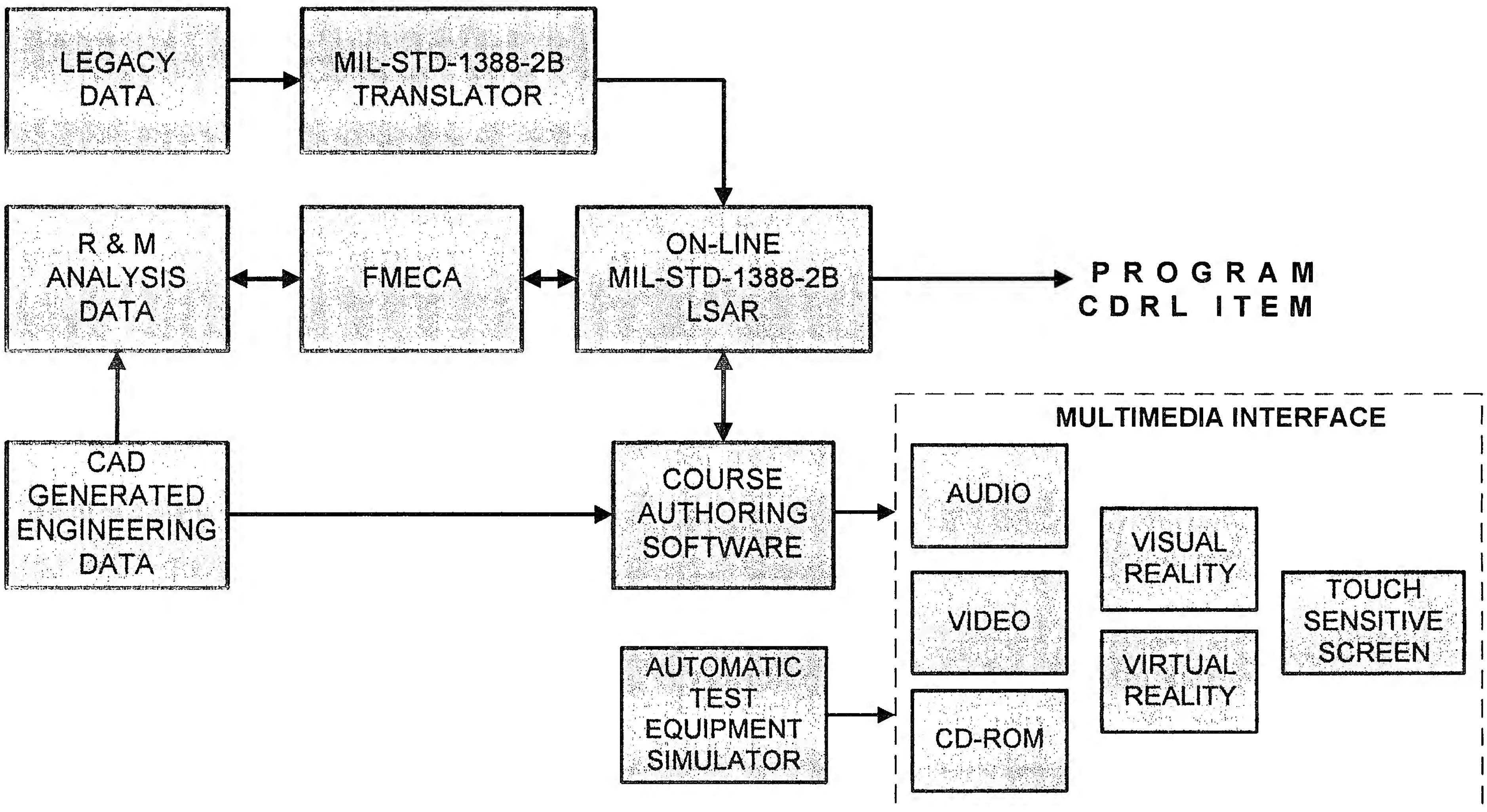
By Carmine Tarantino





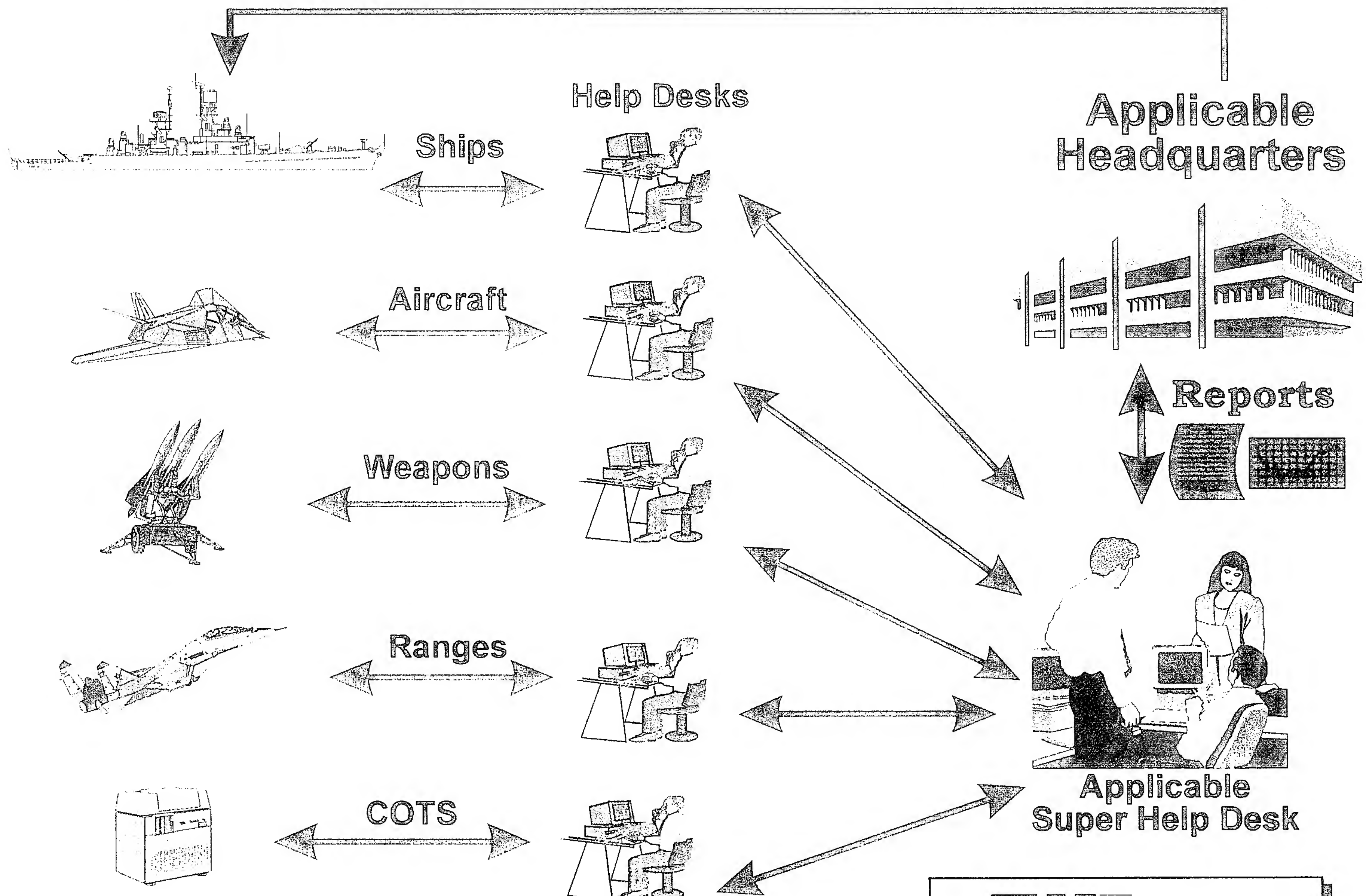
AIL SYSTEMS INC.
TECHNICAL SERVICES OPERATIONS

Dynamic-Portable Automated Customizable Training System (PACTS)



AIL SYSTEMS INC.
TECHNICAL SERVICES OPERATIONS

System Logistics Support & Reporting



AIL SYSTEMS INC.
TECHNICAL SERVICES OPERATIONS

The Super Help Desk Support Team

573

Super Help Desk Team

- ✓ Support Engineers
- ✓ Knowledge Engineers
- ✓ Knowledge Librarians
- ✓ Configuration Management Engineers
- ✓ Maintenance Engineers
- ✓ Logistics Engineers
- ✓ Technical Writers
- ✓ Administrative Support
- ✓ Clerical Support

AIL SYSTEMS INC.
TECHNICAL SERVICES OPERATIONS

571

MULTI - SIGNAL MODELING: A NEW APPROACH FOR SYSTEM TESTABILITY ANALYSIS AND FAULT DIAGNOSIS

Dr. DAVID L. KLEINMAN *

QUALTECH SYSTEMS, Inc.

and

UNIVERSITY OF CONNECTICUT
(Dept. of Electrical and Systems Engrg.)

June 15, 1995

* Currently on leave at Naval Postgraduate School, Monterey, CA

CRITERIA FOR A GOOD MODELING METHODOLOGY FOR FAULT DIAGNOSIS

- MUST BE USABLE FOR DFT IN A CONCURRENT ENGINEERING ENVIRONMENT
  - model should evolve as product evolves from:

schematics ⇒ design of subsystems ⇒ integrated system ⇒ fielded system ⇓ LRU / SRU etc. ⇓ hierarchical maintenance

top down | bottom up ⇓ hierarchical modeling

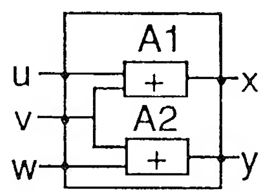
- EASY TO VALIDATE (CLOSE CONFORMANCE TO STRUCTURE)
- EASY TO INTEGRATE SYSTEM MODELS FROM SUBSYSTEM MODELS
- EASY TO BUILD - INTUITIVE METHODOLOGY REQUIRING MINIMAL TIME AND INFORMATION, WHILE PROVIDING HIGH DIAGNOSTIC RESOLUTION

185

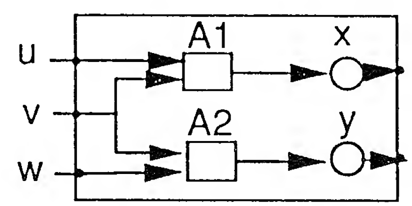
DIGRAPH MODELS FOR SYSTEM TESTABILITY AND FAULT DIAGNOSIS

- DIGRAPH (DEPENDENCY) MODELS CAPTURE CAUSE-EFFECT DEPENDENCIES
  - module nodes ⇒ failure sources
  - test nodes ⇒ monitoring points, observations
  - directed links ⇒ cause-effect relations among module nodes
  - AND nodes ⇒ redundancies for fault-tolerance
  - BIST nodes ⇒ local test in isolation
  - switches ⇒ modes of operation, loop breaking in test mode
- EXAMPLE

ADDER

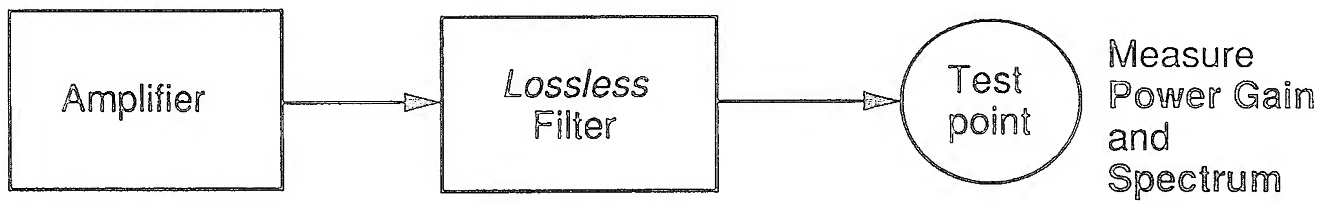


DIGRAPH MODEL



- LINKS DENOTE FIRST-ORDER DEPENDENCIES
  - failure source - test dependencies (e.g., test x detects faults in A1)
  - test x could fail due to upstream failures ⇒ higher-order dependencies

A SIMPLE AMPLIFIER-FILTER EXAMPLE



- QUANTITATIVE AND QUALITATIVE MODELS
  - specify amplifier and filter transfer functions, difference equations,...
  - context sensitive (e.g., for HF circuits, TF = f (Gain-Bandwidth product and slew rate))

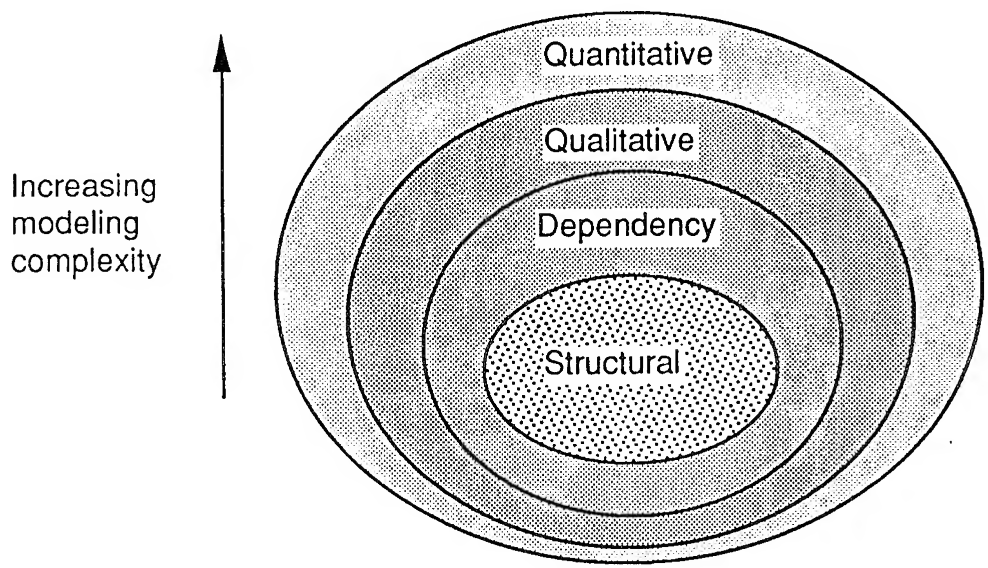
  $\Rightarrow$ require too much information, even for a small system

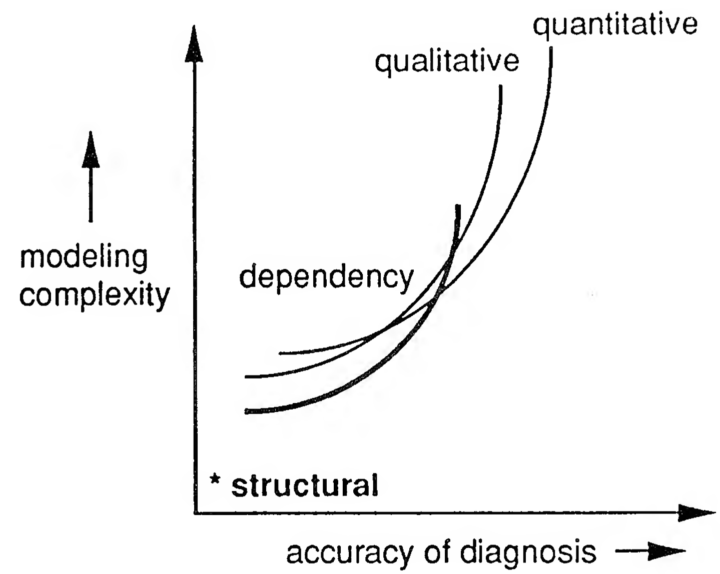
- STRUCTURAL MODELS
  - require minimal information $\Rightarrow$ just the block diagram

  $\Rightarrow$ do not use functional information (e.g., if spectrum is incorrect, the filter must be faulty)

- DEPENDENCY MODELS
  - make use of functional information, but **deviate significantly from structure**

EXISTING MODELING TECHNIQUES FOR FAULT DIAGNOSIS



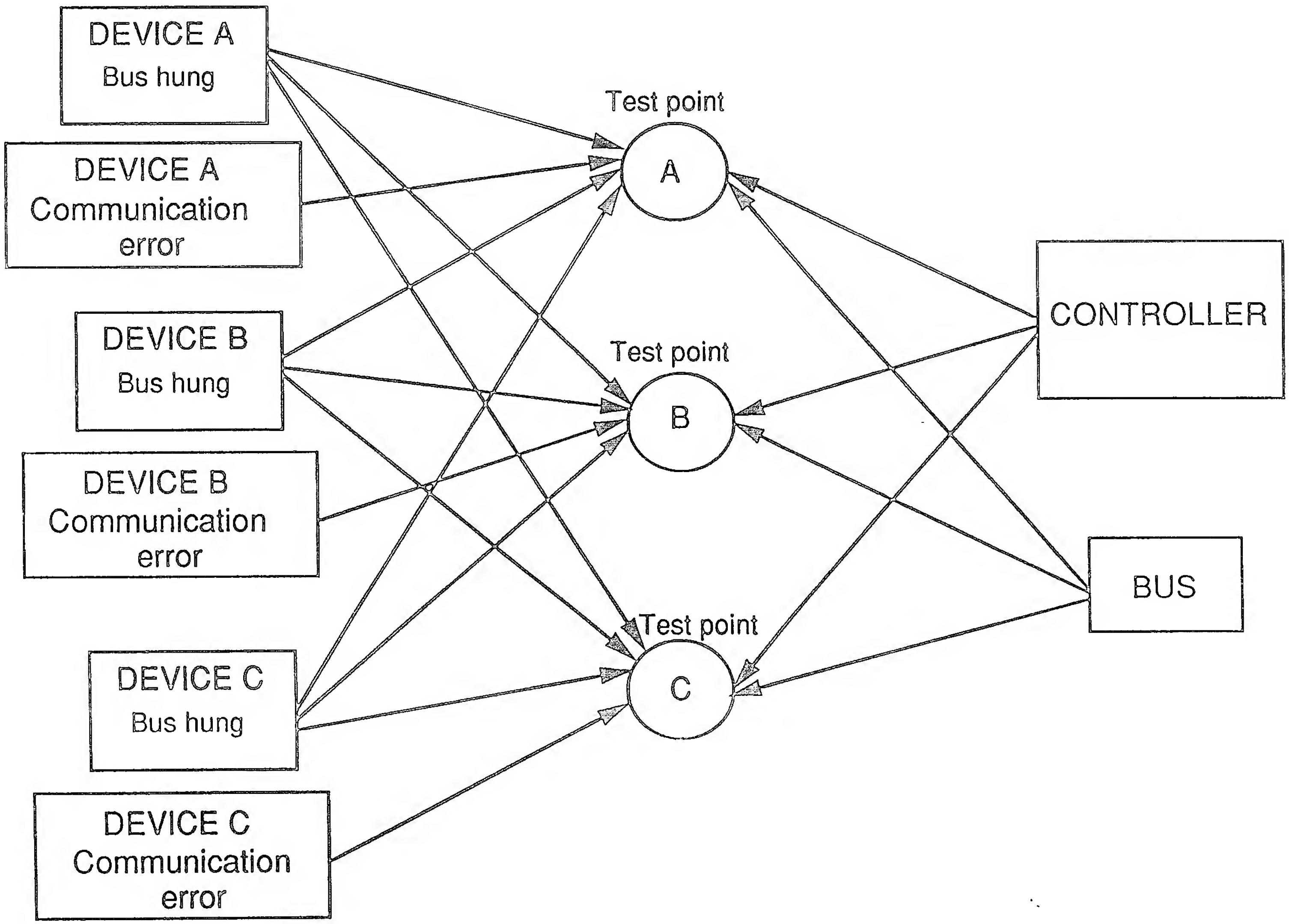


TRADE OFF BETWEEN MODELING COMPLEXITY AND DIAGNOSTIC RESOLUTION

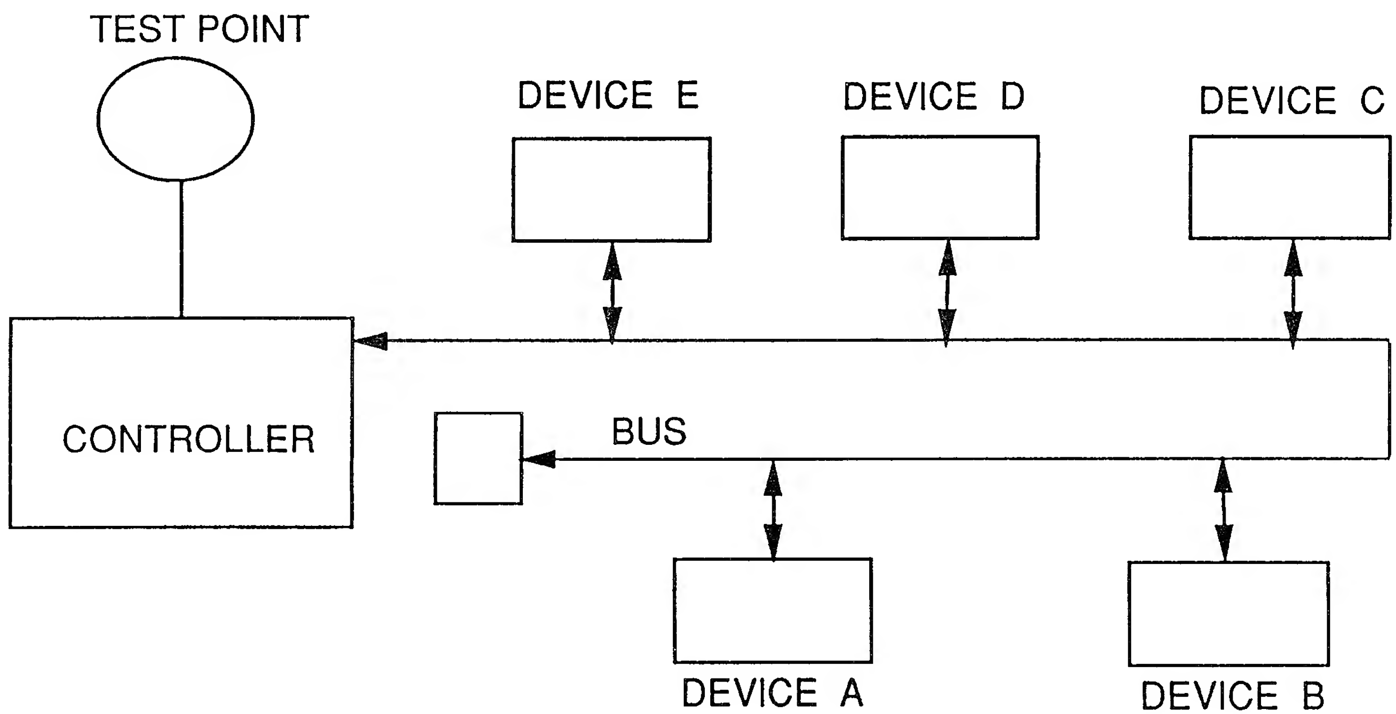
2 seconds to generate diagnostic strategy via dependency modeling using TEAMS versus 3.5 hours by a qualitative reasoning system for a helicopter blade de-ice system

583

DEPENDENCY MODEL OF A SIMPLE BUS SYSTEM (for three devices only)

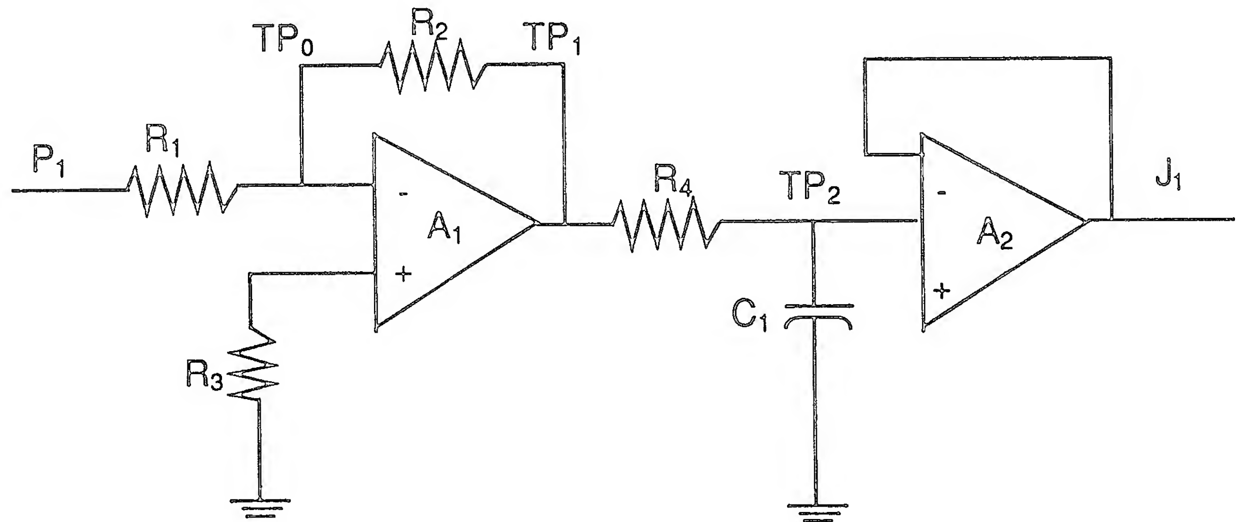


A SIMPLE BUS SYSTEM

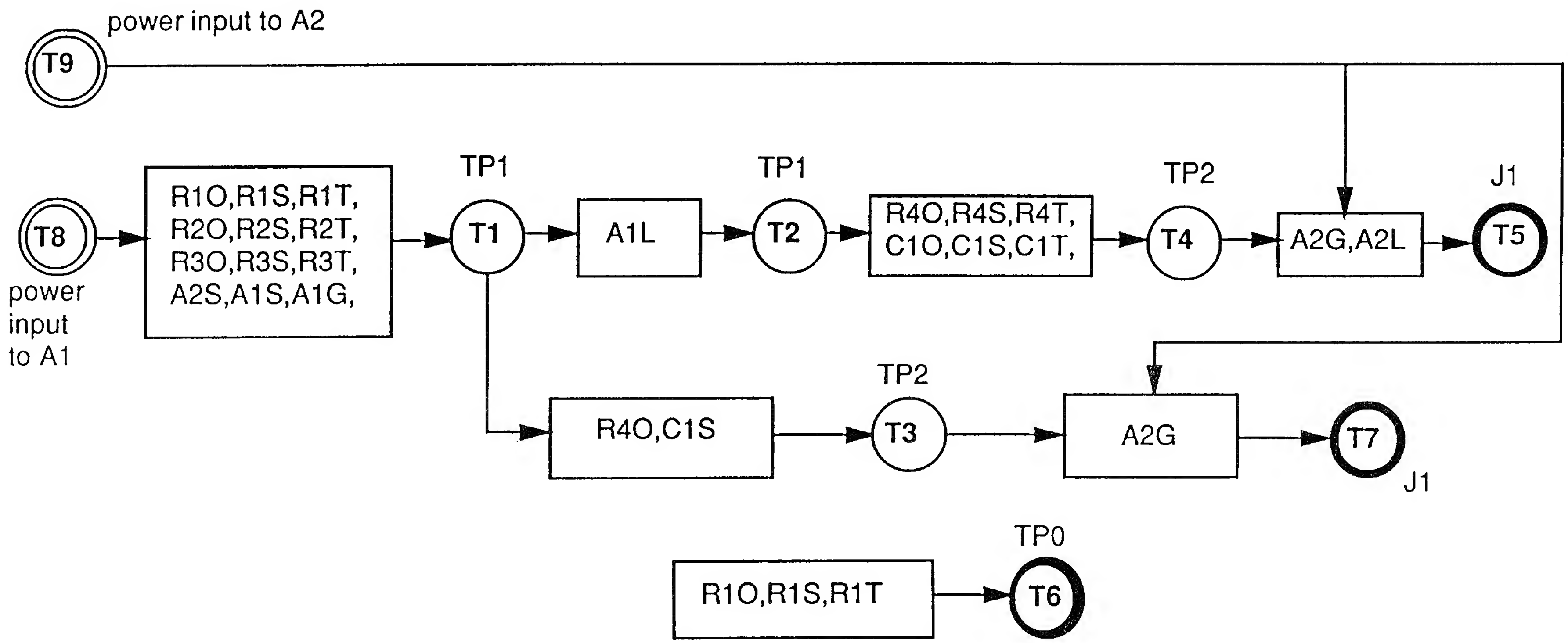


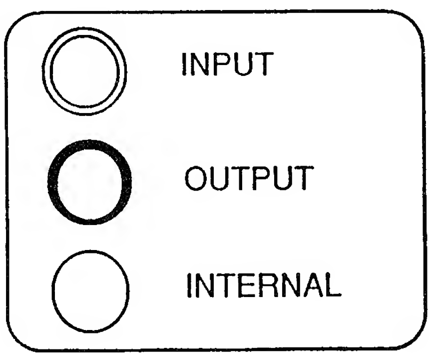
- TEST $\Rightarrow$ SEND A QUERY TO EACH DEVICE TO ASCERTAIN ITS STATUS

AN AMPLIFIER / FILTER SYSTEM



DEPENDENCY MODEL OF AMPLIFIER - FILTER EXAMPLE





Legend : O - open
S - short
T - out-of-tolerence

T1, T3,T7 : dc tests
T2,T4,T5 : ac tests
T6: measure R1
T8,T9: apply power

Source : WSTA Training Course, June 1993

TWO TYPES OF FAILURES AND HOW THEY ARE MODELED

- FUNCTIONAL FAILURE
  - a component fails to perform its function properly (e.g., Q-factor of filter is not high enough due to resistance in the inductor)
- GENERAL FAILURE
  - a catastrophic failure affecting attributes beyond its normal functioning (e.g., a short-circuit in the "*lossless*" filter causes the output power to be zero)

	FUNCTIONAL FAILURES	GENERAL FAILURES
QUANTITATIVE QUALITATIVE	functioning of the system is modeled in great detail ⇒ exceptional capabilities in handling functional failures	modeled as a special case (constraint suspension)
STRUCTURAL	ignored	only failure type considered
DEPENDENCY MODELING	handled via "failure modes" ⇒ structural distortion ⇒ failure modes are "pseudo-components"exhibiting general failures only	default failure type

689

HOW DO EXISTING MODELING METHODOLOGIES COMPARE ?

	QUANTITATIVE	QUALITATIVE	DEPENDENCY	STRUCTURAL
DFT & CE	*	*	Good	Excellent
Validation	Fair	Fair	Poor	Excellent
Integration	Good	Good	Fair	Excellent
Ease of modeling	Extremely Poor	Poor	Fair	Excellent
Diagnostic accuracy	Excellent	Good	** Good / Fair	Poor
Modeling cost	Prohibitively Expensive	Very Expensive	Moderate	Cheap

* typically not available during early design phases
** varies widely, depending on skills and efforts of modeler

GOAL : Develop a modeling methodology that has the ease of structural modeling, and the accuracy of dependency modeling or better $\Rightarrow$ Multi-signal modeling

RATIONALE FOR MULTI-SIGNAL MODELING

- COMPONENT DEPENDENCIES EXIST *ONLY IF* THERE IS A CONNECTING PATH
  - the links in a structural model identify the paths
  - each path is a potential dependency
- A PATH *DOES NOT* ALWAYS IMPLY DEPENDENCY
  - example : addressable devices on a bus
  - structural models *ignore* this possibility
- DEPENDENCIES ARE *MULTI-DIMENSIONAL*
  - the connecting path carries complex attribute information
  - each component alters one or more attributes(e.g., amplifier alters power,filter alters spectrum)
  - dependencies exist if one or more attributes are shared by components/tests
  - structural models ignore functional attribute information
  - dependency models include attribute information via failure modes $\Rightarrow$ structural distortion
- NOT SIMULATING THE NORMAL FUNCTIONING OF THE SYSTEM
  - significantly reduced modeling cost $\Rightarrow$ can solve larger modeling problems

FORMAL DEFINITION OF A MULTI-SIGNAL MODEL

- A MULTI-SIGNAL MODEL CONSISTS OF :
  - a finite set of components $C = \{c_1, c_2, \ldots\ldots\ldots, c_L\}$ and a set of independent signals $S = \{s_1, s_2, \ldots\ldots\ldots, s_K\}$ associated with the system
  - a finite set of n available tests $T = \{t_1, t_2, \ldots\ldots\ldots, t_n\}$
  - a finite set of P available test points (or probe points) $TP = \{TP_1, TP_2, \ldots\ldots\ldots, TP_P\}$
  - a finite set of operations (preconditions) O associated with tests **(optional)**
  - a finite set of resources R associated with system testing **(optional)**
  - each test point TP_P is associated with a subset of tests, $SP(TP_P)$
  - each component c_i affects a subset of signals, $SC(c_i)$
  - each test t_j checks a subset of signals $ST(t_j)$
  - each test t_j requires a subset of operations to be performed, $OP(t_j)$ **(optional)**
  - each operation o_p requires a subset of resources $RS(o_p)$ **(optional)**
  - the digraph $DG = \{C, TP, E\}$, where E denotes the set of directed edges specifying the *structural connectivity* of the system

Dependency model = Muti-signal model with K=1 and DG = dependency graph

A THREE-STEP GUIDE TO MULTI-SIGNAL MODELING - 1

- ENTER THE STRUCTURAL MODEL, SCHEMATIC MODEL OR A CONCEPTUAL BLOCK DIAGRAM
  - the structural model can be generated from VHDL structural models, EDIF netlists or via the graphical user interface

- ADD SIGNALS TO THE MODULES AND TESTS
  - identify the set of signals from functional specifications or from the independent variables in the transfer function.

    example : signal specification of a power amplifier will include output distortion, harmonic distortion and power output
  - any unique attribute will have an associated signal

    example : in a bus with multiple independently addressable devices, the address of each device will serve as a signal

573

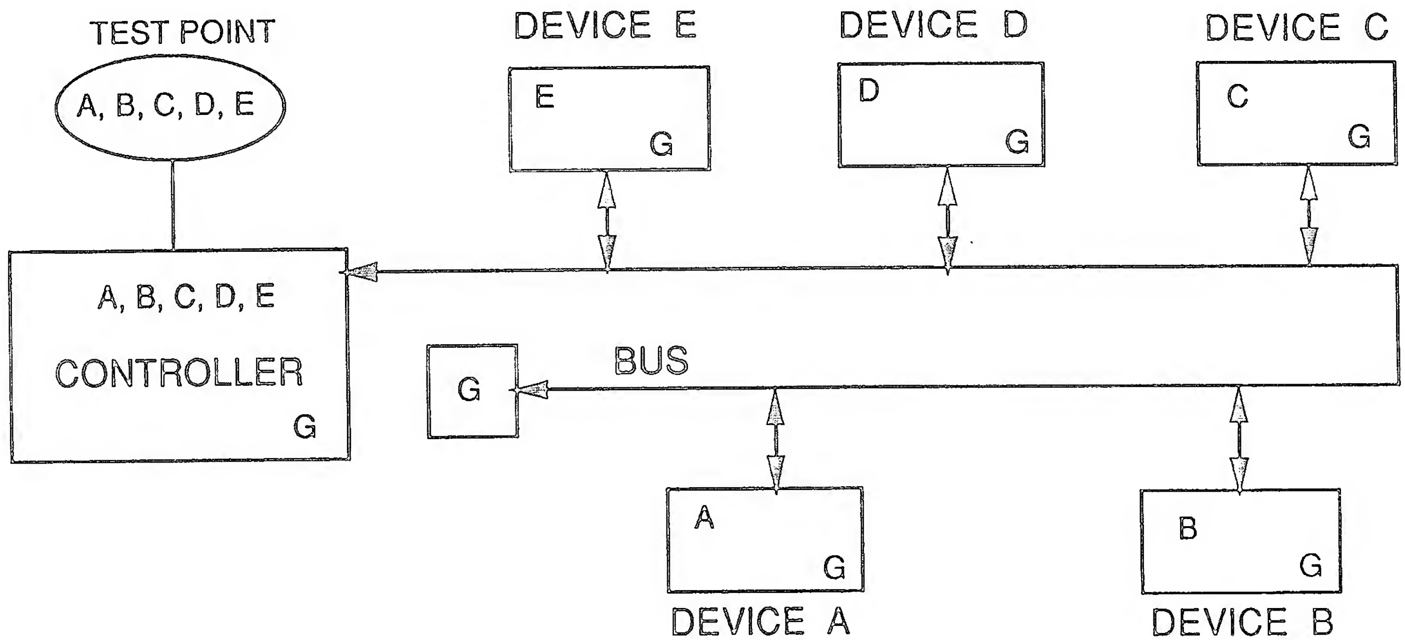
A THREE-STEP GUIDE TO MULTISIGNAL MODELING - 2

- UPDATE MODEL FOR SPECIAL SITUATIONS
  - for a built-in redundant system (e.g., both A&B must fail for a system to fail), configure the redundant components using *AND* nodes
  - for a system having different modes of operation, use special purpose nodes called SWITCHES to model them
  - if a system has components with built-in -self-test (BIST), enable BIST in property options of the component in the TEAMS model.† BIST can also be modeled via unique signal names. A BIST spanning multiple components is also called a "component test" in dependency modeling; it can be modeled via a unique signal
  - if a system has replaceable digital integrated circuits, model them with their equivalent models. The equivalent models are simplified models of chips that capture the necessary dependency information to detect faults in (but not isolate faults within) a chip

TESTABILITY ENGINEERING AND MAINTENANCE SYSTEM (TEAMS) INTEGRATES MULTI-SIGNAL MODELING METHODOLOGY AND THE ASSOCIATED ALGORITHMS IN AN EASY-TO-USE GRAPHICAL INTERFACE

† TEAMS is a software tool developed by QUALTECH SYSTEMS, Inc., Storrs, CT 06268

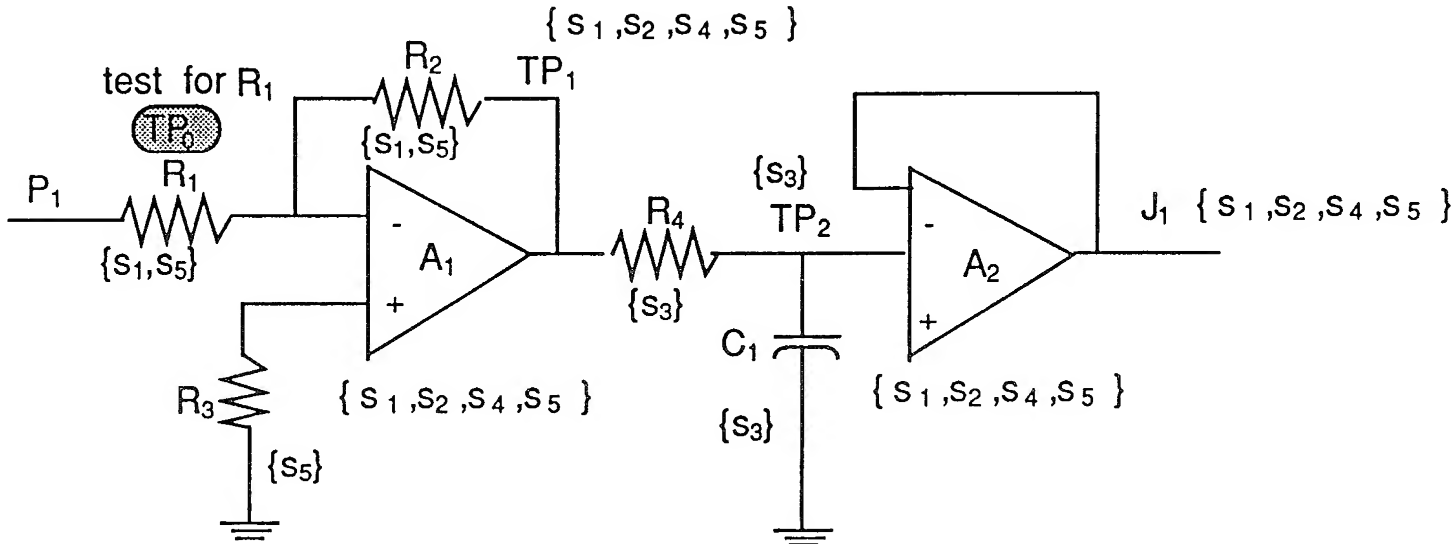
A MULTI-SIGNAL MODEL OF A SIMPLE BUS SYSTEM



Legend: G $\Rightarrow$ general failure
A-E $\Rightarrow$ signals for individually addressable devices

- NOTE THE SIMILARITY TO STRUCTURE
- ANALYSIS BASED ON STRUCTURAL MODEL ALONE WILL RESULT IN ALL DEVICES AND CONTROLLER BEING IN ONE AMBIGUITY GROUP

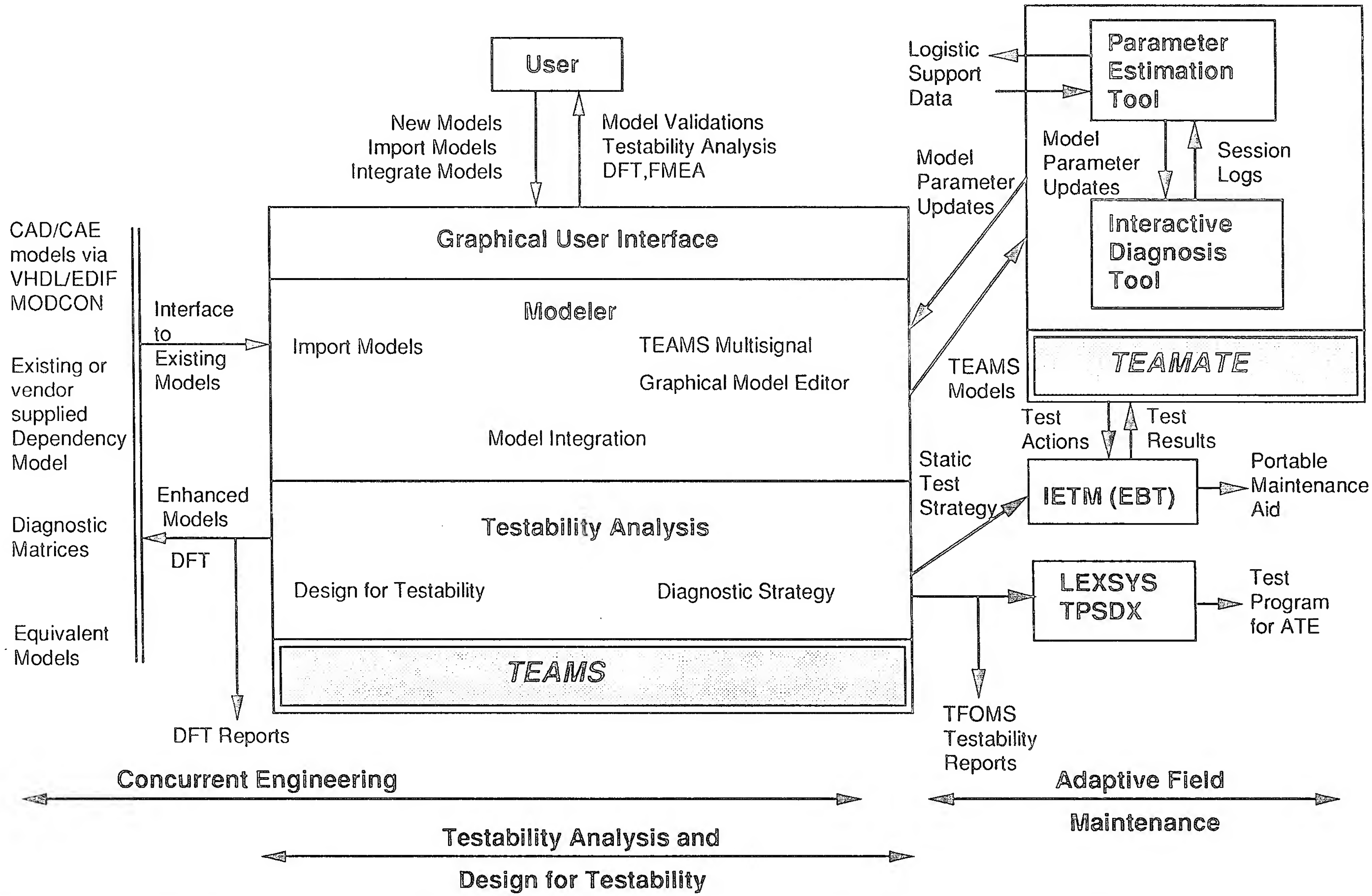
MULTISIGNAL DEPENDENCY MODEL OF AN AMPLIFIER / FILTER



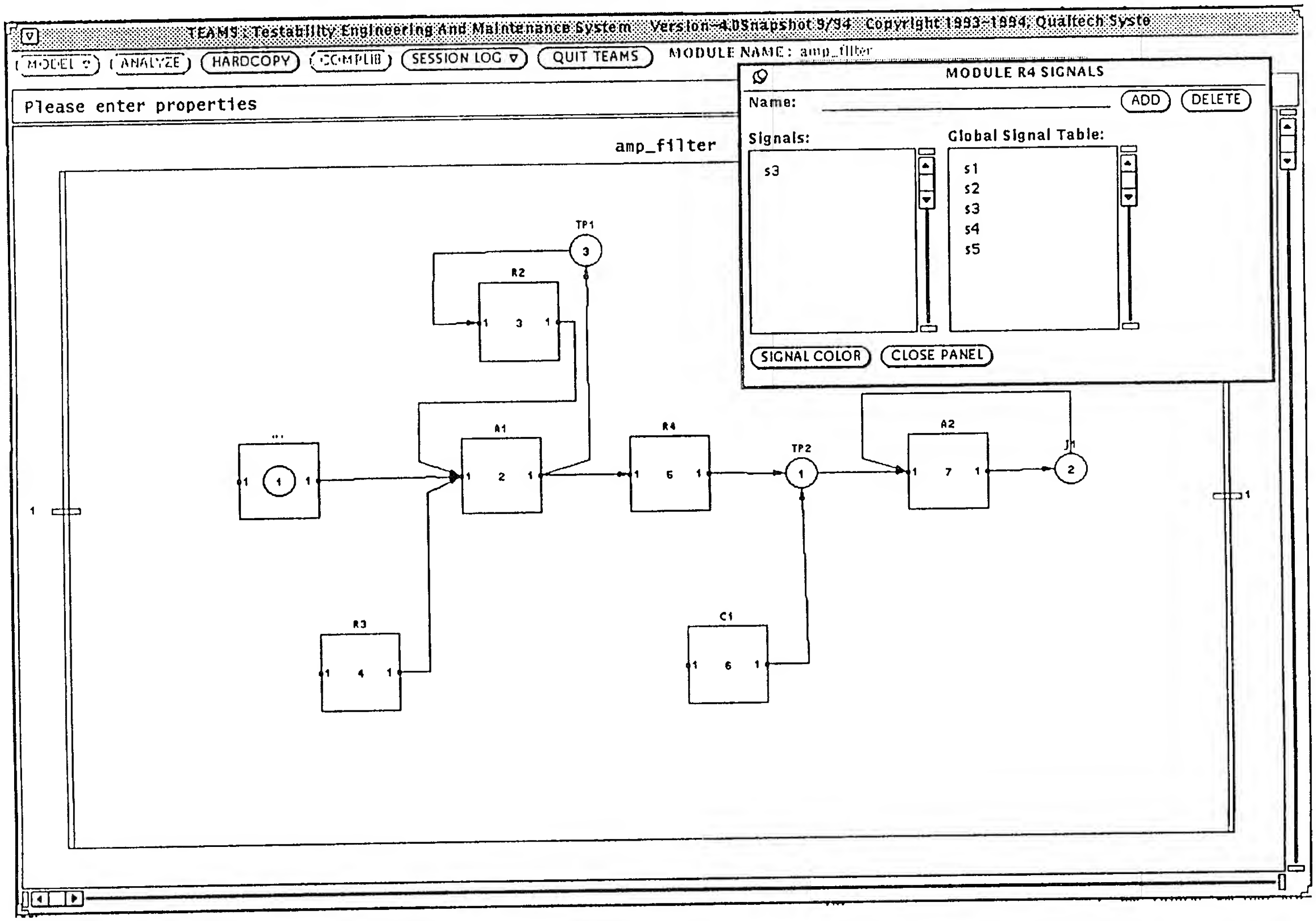
Legend: $s_1 \Rightarrow$ gain ; $s_2 \Rightarrow$ linearity ; $s_3 \Rightarrow$ cut-off frequency ;
$s_4 \Rightarrow$ slew rate ; $s_5 \Rightarrow$ d.c. offset
TP0 : separate test for R_1

565

TEAMS : Integrated Toolset + Reusable Models = Complete Solution

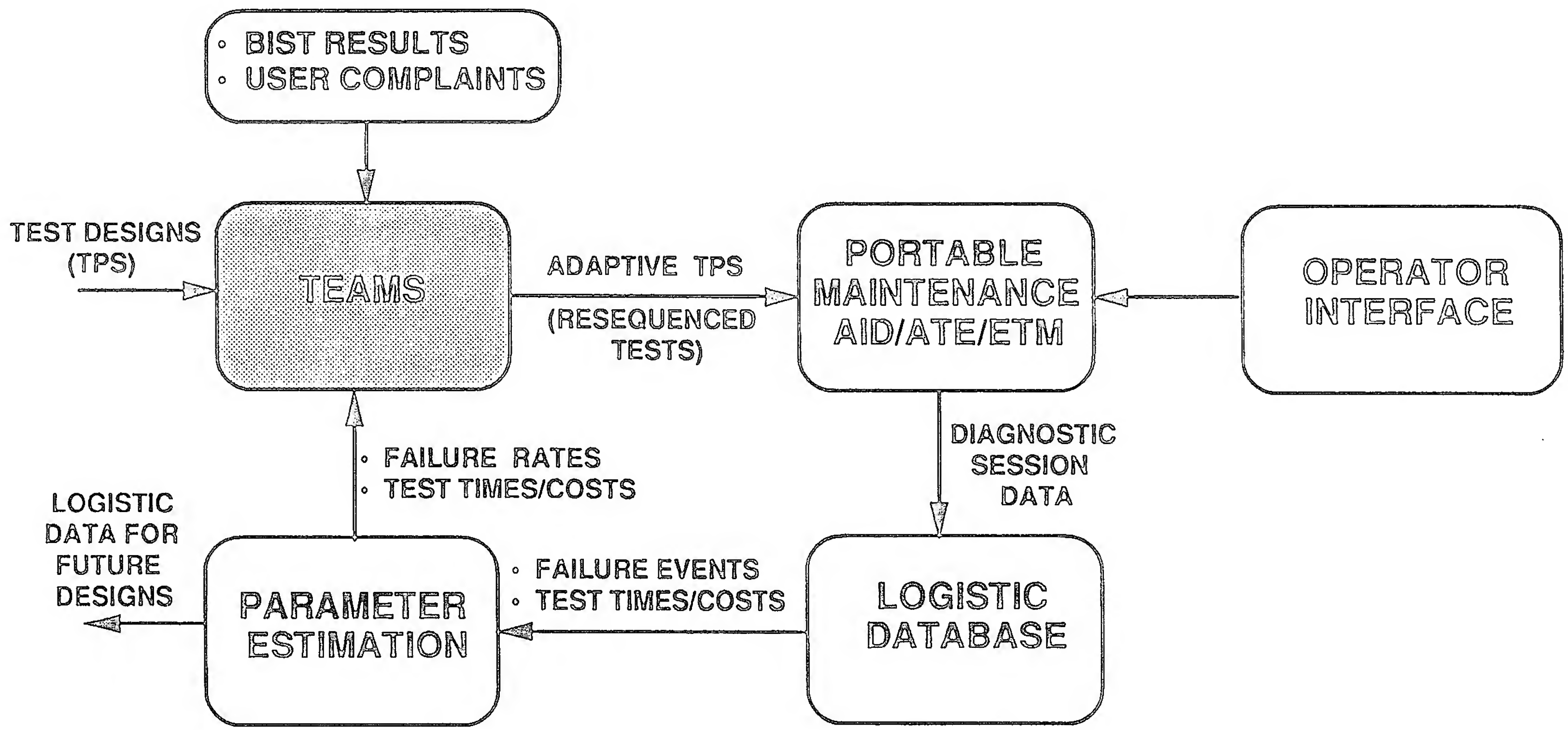


MULTISIGNAL MODEL OF AN AMPLIFIER / FILTER SYSTEM IN TEAMS



597

ROLE OF TEAMS IN INTELLIGENT MAINTENANCE AIDING



- SEQUENCE TESTS ADAPTIVELY ON THE BASIS OF MAINTENANCE HISTORY
- CAN ALSO BE USED FOR OPERATOR TRAINING

597

CONCLUSIONS

- BENEFITS OF MUTI-SIGNAL MODELING
  - **close conformity to structure** ⇒ model creation,validation and integration is easier
  - high diagnostic resolution without the complexity of qualitative or quantitative modeling ⇒ significantly reduced modeling cost
  - **modeling of advanced testability features without deviating from structure**
    - multiple tests on a test point, component tests, test groups,....
  - model **usable** in all phases of a system life-cycle
- TEAMS CAN HANDLE REAL WORLD FEATURES SUCH AS
  - rectification
  - modular diagnosis,
  - precedence restrictions
  - setup operations, test precedences and resource constraints
  - unreliable tests
  - multiple fault isolation

MULTI-SIGNAL MODELING IS A NATURAL NEXT STEP FOR and
A MAJOR ADVANCE OVER DEPENDENCY MODELING

UTILIZATION OF INTELLIGENT DIAGNOSTICS TO REDUCE THE RISK OF G/COTS

(GOVERNMENT/COMMERCIAL OFF-THE-SHELF)

WITH APPLICATION TO :

AVIONICS INTEGRATION AND MODIFICATION

William J. Lucas
North American Aircraft Modification Division
and
Richard K. Walker, Don R. Allen & James J. Hoote
Diagnostic Consultants

109

A NEW WAY OF DOING BUSINESS

(SecDef Memo of 29 Jan 1994, subj: Specifications & Standards - A New Way of Doing Business)

- ✓ NEED COST EFFECTIVE DEFENSE SYSTEMS
- ✓ DECREASE RELIANCE ON MILITARY SPECIFICATIONS AND STANDARDS
- ✓ INCREASE ACCESS TO COMMERCIAL TECHNOLOGY

COTS CONSIDERATIONS

(From NSIA/DoD Meeting, June 1994)

- DESIGN SPECIFICATIONS
  - DIFFERENT FROM MIL-STD
  - DIFFERENT VERIFICATIONS
- DATA
  - WHAT YOU PAY FOR
- SUPPORT
  - AT VENDOR'S MERCY
- COST
  - SUBSTANTIALLY LOWER (INITIALLY)
  - MODS ARE EXPENSIVE
- LONG TERM AVAILABILITY
  - AT VENDOR'S MERCY
- INTERCHANGEABILITY
  - NO REQUIREMENT

603

CUSTOMER SELECTION CRITERIA

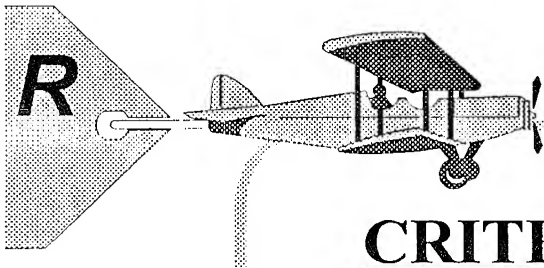
√ COST

√ RISK

√ PERFORMANCE

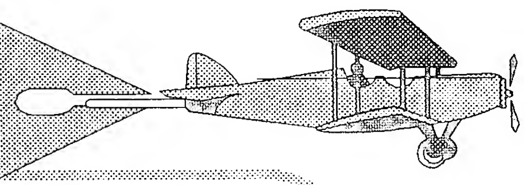
√ SUPPORTABILITY

COTS PARADIGM



CRITERIA		CUSTOMER PERCEPTION*
√ COST	→	LOW (Substantially)
√ RISK	→	HIGH (Detailed design unknown)
√ PERFORMANCE	→	ADEQUATE (COTS not equal to MIL-STD)
√ SUPPORTABILITY	→	POOR (Varies with product and supplier)

COTS PARADIGM

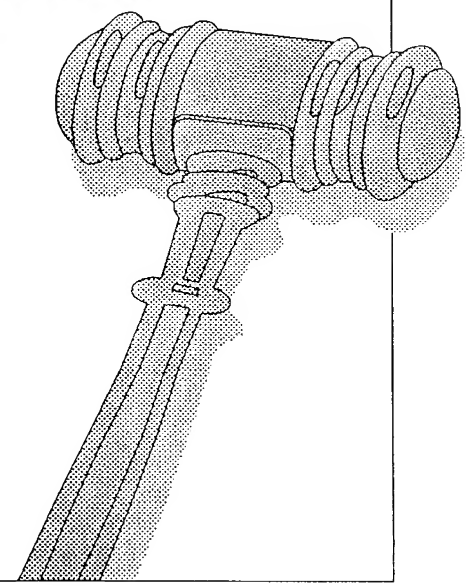


*WE CAN IMPROVE CUSTOMER ACCEPTANCE OF COTS

CRITERIA		CUSTOMER PERCEPTION*
√ COST	→	LOW (Substantially)
√ RISK	→	HIGH (Detailed design unknown)
√ PERFORMANCE	→	ADEQUATE (COTS not equal to MIL-STD)
√ SUPPORTABILITY	→	POOR (Varies with product and supplier)

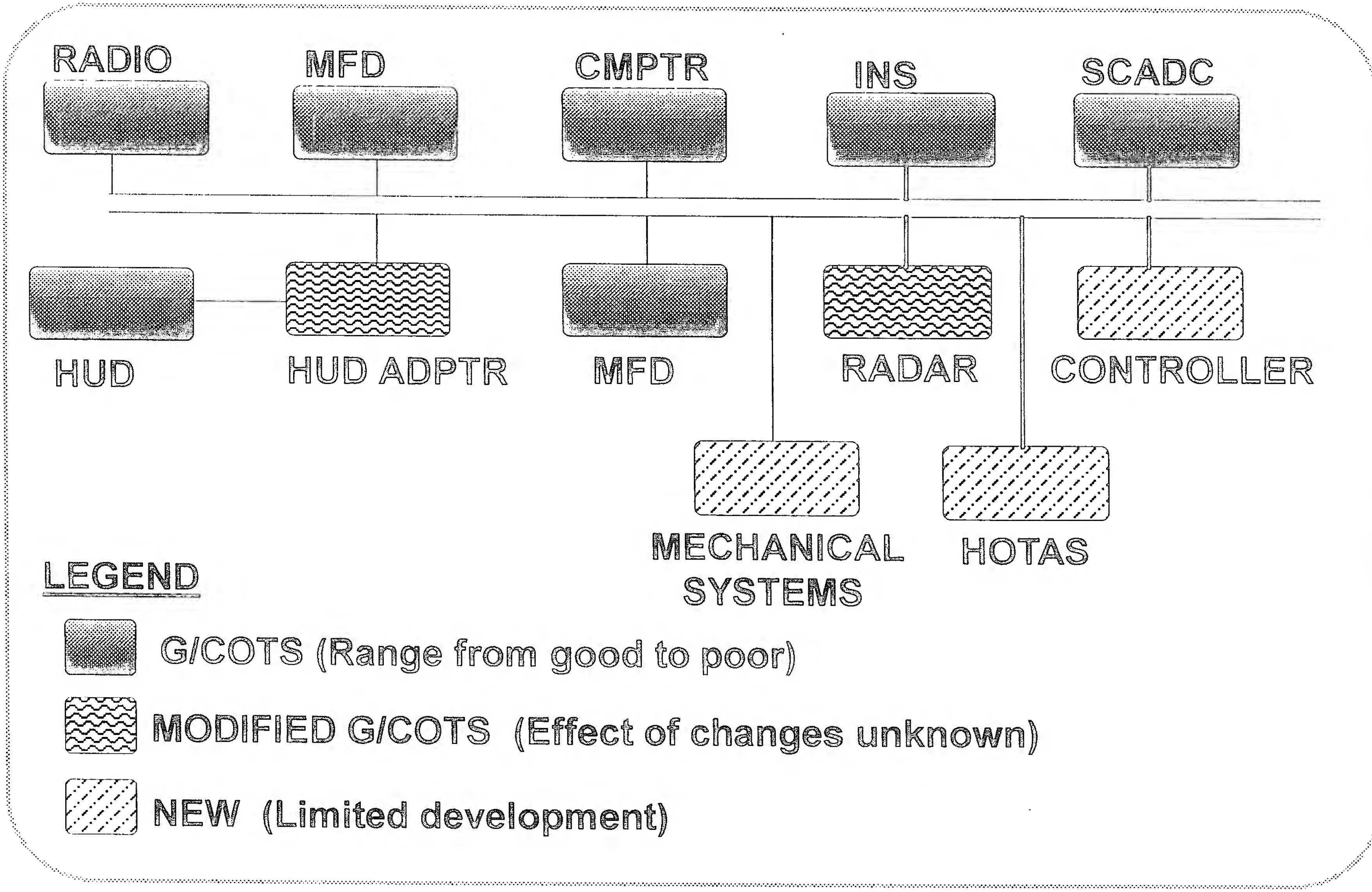
THE REALITY

A MIX OF GOVERNMENT/COMMERCIAL
OFF-THE -SHELF (G/COTS)
WITH LIMITED OPPORTUNITY
FOR DEVELOPMENT AND
MODIFICATION



NOTIONAL ARCHITECTURE

A MIX OF G/COTS SYSTEMS WITH LIMITED DEVELOPMENT AND MODIFICATION



TYPICAL AVIONICS INTEGRATION

- IDENTIFY NECESSARY PERFORMANCE ENHANCEMENTS
  - PROVIDE AT LOWEST POSSIBLE PRICE
  - PROVIDE ONLY ESSENTIAL SUPPORT CONSIDERATIONS
- SELECT APPROPRIATE SYSTEM VENDORS
  - GOVERNMENT/COMMERCIAL-OFF-THE-SHELF (G/COTS) SYSTEMS
  - LIMIT MODS TO ABSOLUTE MINIMUM
- DEVELOP OPERATIONAL FLIGHT PROGRAM (INCLUDING BIT INTEGRATION) AND VERIFY PERFORMANCE IN LAB
- MODIFY PROTOTYPE VEHICLE AND FLIGHT TEST
- PROVIDE KITS AND INSTALLATION AS REQUIRED
- PROVIDE SUPPORTABILITY FUNCTIONS AS REQUIRED

TYPICAL DIAGNOSTICS APPROACH

(CONSTRAINED TO LIMIT DEVELOPMENT COSTS)

EVENT		PLAN
◇ VEHICLE/SYSTEM DIAGNOSTIC SPECIFICATIONS	→	NONE PLANNED
◇ VEHICLE BIT SPECIFICATION - REVIEW VENDOR SPECIFICATION - CREATE NEW SPEC AS REQUIRED	→	NO FD/FI REVIEW/INPUT
◇ BIT INTEGRATION ACTIVITY - DEVELOP/VERIFY VEHICLE BIT SOFTWARE	→	NO FD/FI VERIFICATION
◇ BIT ACTIVITY DURING FLIGHT TEST - IDENTIFICATION/CORRECTION OF BIT ERRORS	→	NO FD/FI VERIFICATION
◇ COTS RISK MITIGATION REPORT	→	NONE IDENTIFIED

INCREASED RISK

- UNDETECTED FAULTS
- LARGE AMBIGUITY GROUPS
- UNNECESSARY TESTS
- MASKING FALSE FAILURES
- FALSE ALARMS
- FEEDBACK LOOPS

INCREASED RISK IMPACT

- EXCESSIVE TIME/ COST TO ISOLATE FAULTS
- EXCESSIVE TIME/COST TO REPAIR FAULTS
- UNNECESSARY TESTING
- DETERIORATED MAINTENANCE (e.g.: Swaptronics and Cannibalization)

INCREASED RISK IMPACT

(SYSTEMS INTEGRATION LABORATORY)

PROBLEM	RESULT
FAILURE TO DETECT BEFORE INTEGRATION →	FAULTY TEST DATA UNNECESSARY TESTING
FAILURE TO DETECT DURING INTEGRATION →	INVALID TEST LOST TEST TIME
FAULTY ISOLATION AND REPAIR →	SYSTEM UNAVAILABILITY EXCESSIVE LAB MANHOURS

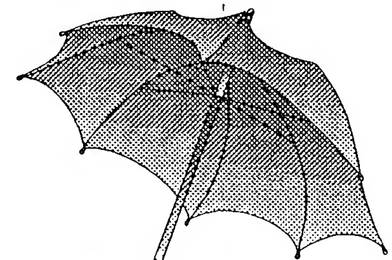
613

INCREASED RISK IMPACT

(GROUND BUILD UP/INSTALLATION AND FLIGHT TEST)

PROBLEM		RESULT
FAILURE TO DETECT BEFORE FLIGHT	→	INVALID TEST FLIGHTS
CANNOT DUPLICATE RETEST OK FALSE ALARMS	→	HIGH DOWNTIME HIGH MAINTENANCE HRS EXCESSIVE SPARES LONG REPAIR QUEUE
FAILURE TO DETECT DURING FLIGHT	→	FAULTY FLIGHT DATA INVALID TEST FLIGHTS LOST AIRCRAFT

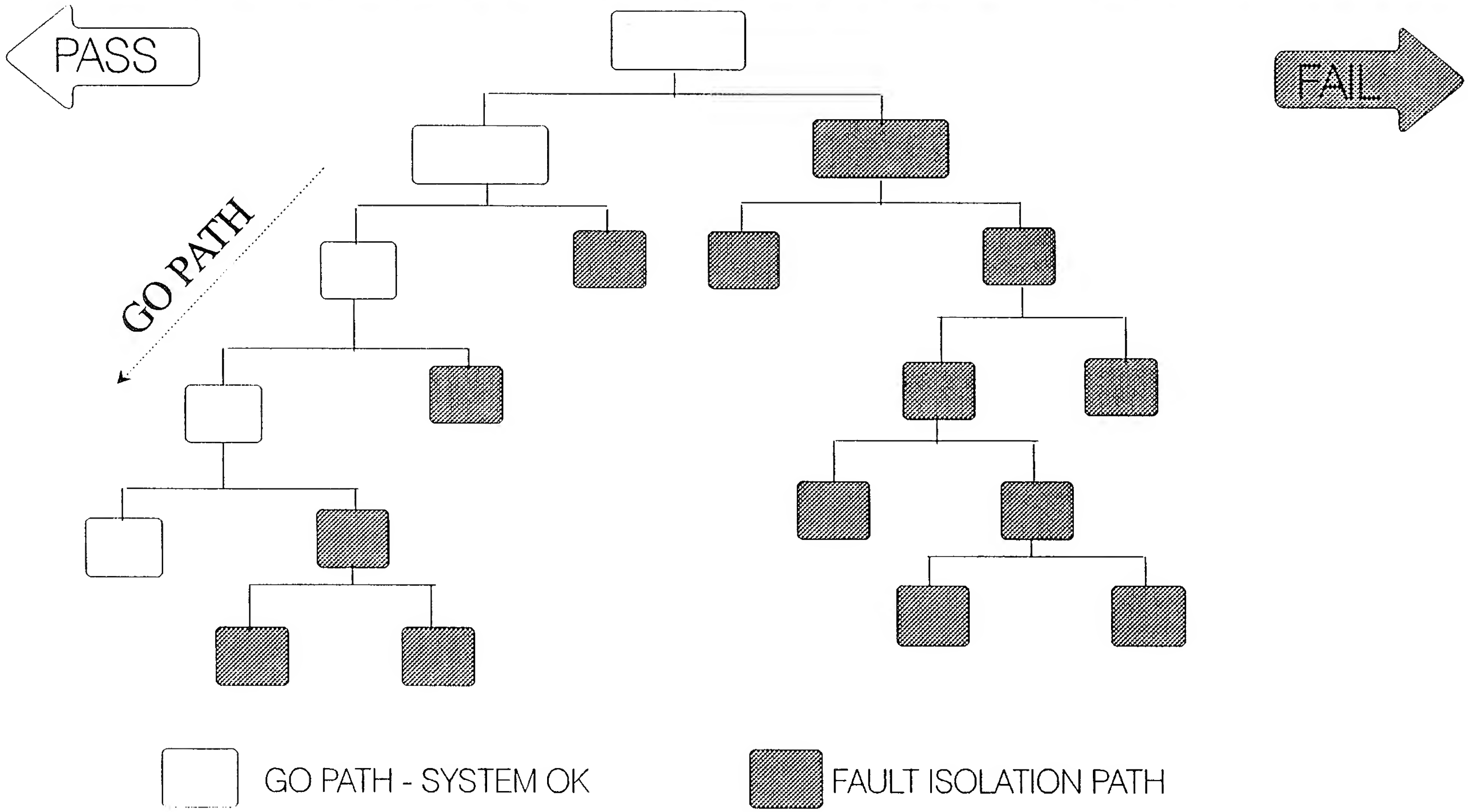
519



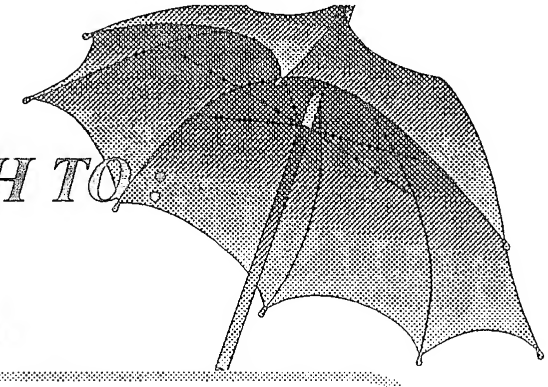
INTELLIGENT DIAGNOSTICS APPROACH TO:

G/COTS RISK MITIGATION

FUNCTIONAL AND DETAILED DEPENDENCY MODELING



INTELLIGENT DIAGNOSTICS APPROACH TO:



G/COTS RISK MITIGATION

(1) REVIEW G/COTS SYSTEMS FD/FI COVERAGE
- COMPARE TO SYSTEM REQUIREMENTS

(2) PROVIDE STRATEGY FOR VEHICLE BIT
- VEHICLE/SYSTEMS DIAGNOSTICS SPEC
- INTEGRATION STRATEGY FOR APPLICABLE LEVELS

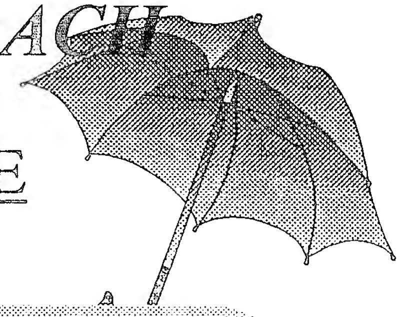
(3) DETERMINE IMPACT OF THE FOLLOWING ON SYSTEMS INTEGRATION AND FLIGHT TEST ACTIVITIES:
- UNDETECTED /HIDDEN FAULTS
- FAULT ERRORS
- SPARES
- MEAN DOWNTIME
- REPAIR QUEUE SIZE

INTELLIGENT DIAGNOSTICS APPROACH

SPECIFIC RISK MITIGATION TECHNIQUE

- **BASED ON CUSTOMER SELECTION CRITERIA**
- **MINIMIZE RISK VS COST**
  - **ASSIGN RELATIVE COST TO SUBSYSTEMS**
  - **ASSIGN RELATIVE RISK TO SUBSYSTEMS**
    - **SAFETY CRITICAL**
    - **MISSION CRITICAL**
    - **AVAILABILITY**
    - **RELIABILITY**
    - **SUPPORTABILITY**

INTELLIGENT DIAGNOSTICS APPROACH



SPECIFIC RISK MITIGATION TECHNIQUE

(SIX STEP IMPLEMENTATION)

- ASSIGN RELATIVE $ COST FOR TOTAL EFFORT
- ESTABLISH RELATIVE $ COST FOR EACH SUBSYSTEM
- ESTABLISH RELATIVE RISK FOR EACH SUBSYSTEM
- DEVELOP ALGORITHM TO DETERMINE RELATIVE EFFORT
  - COST EQUATION
  - RISK EQUATION
- COMPARE SUBSYSTEM DIAGNOSTIC LEVELS
  - EXISTING LEVELS
  - ANALYSIS RESULTS
- REASSIGN DIAGNOSTIC EFFORT

INTELLIGENT DIAGNOSTICS BENEFITS

UP TO 30% AVAILABILITY INCREASE

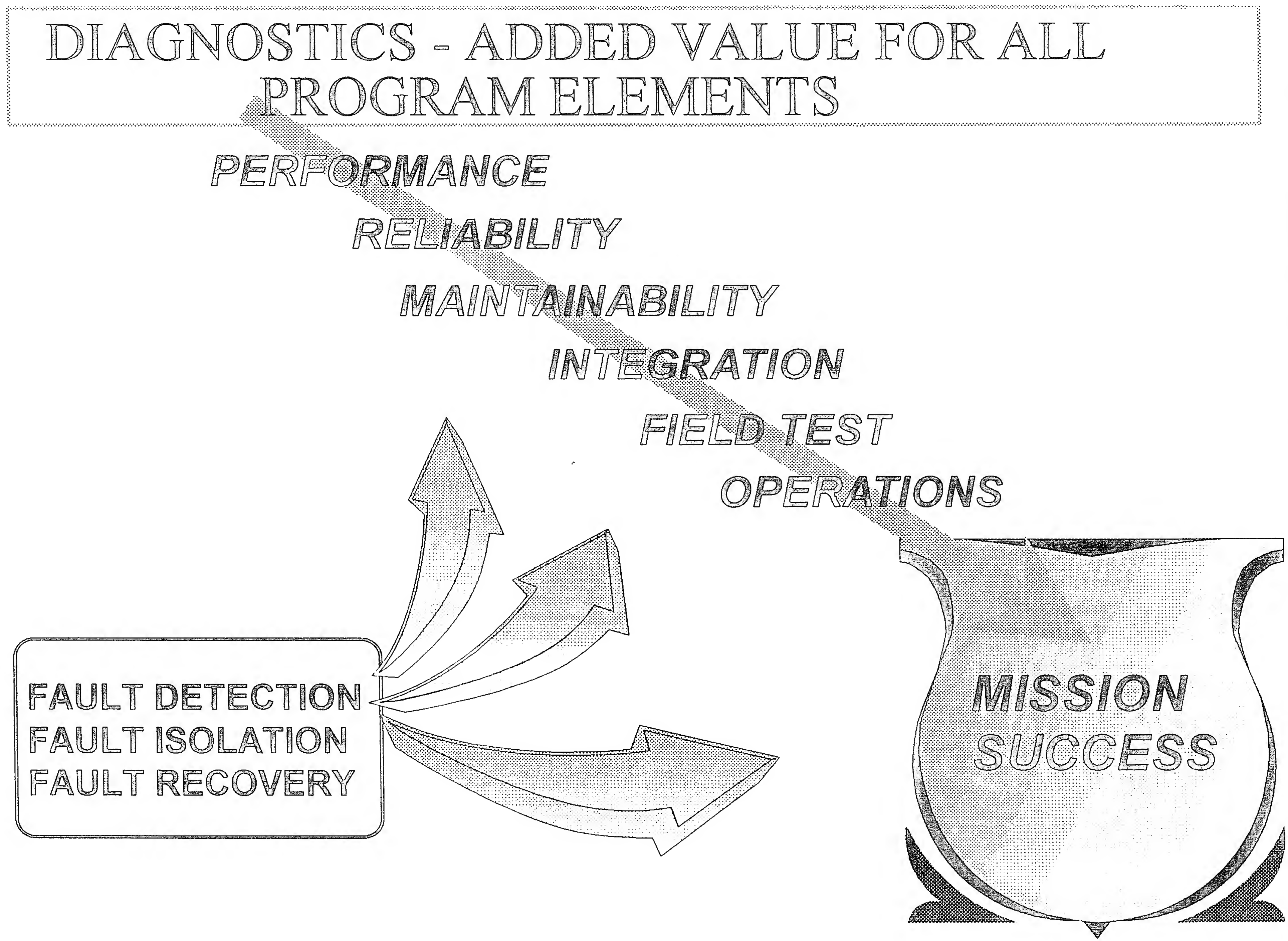
UP TO 60% REDUCTION IN DIAGNOSTICS ACQUISITION COST

UP TO 30% REDUCTION IN CND's & RETOK's

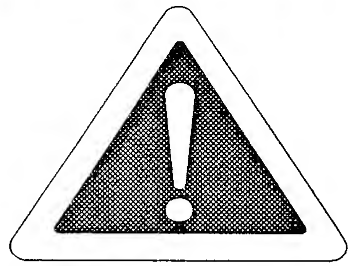
10% REDUCTION IN DOCUMENTATION

30% REDUCTION IN BIT

617


DIAGNOSTICS - ADDED VALUE FOR ALL PROGRAM ELEMENTS
PERFORMANCE
RELIABILITY
MAINTAINABILITY
INTEGRATION
FIELD TEST
OPERATIONS
FAULT DETECTION
FAULT ISOLATION
FAULT RECOVERY
MISSION
SUCCESS

CAUTION



- HARDER THAN IT LOOKS
- REQUIRES DEDICATION
- MUST TRANSLATE GENERIC TO SPECIFIC
- COMMON SENSE RULES
- TRADES ESSENTIAL
- WILL PROVIDE UNIFORMITY AND COST SAVINGS

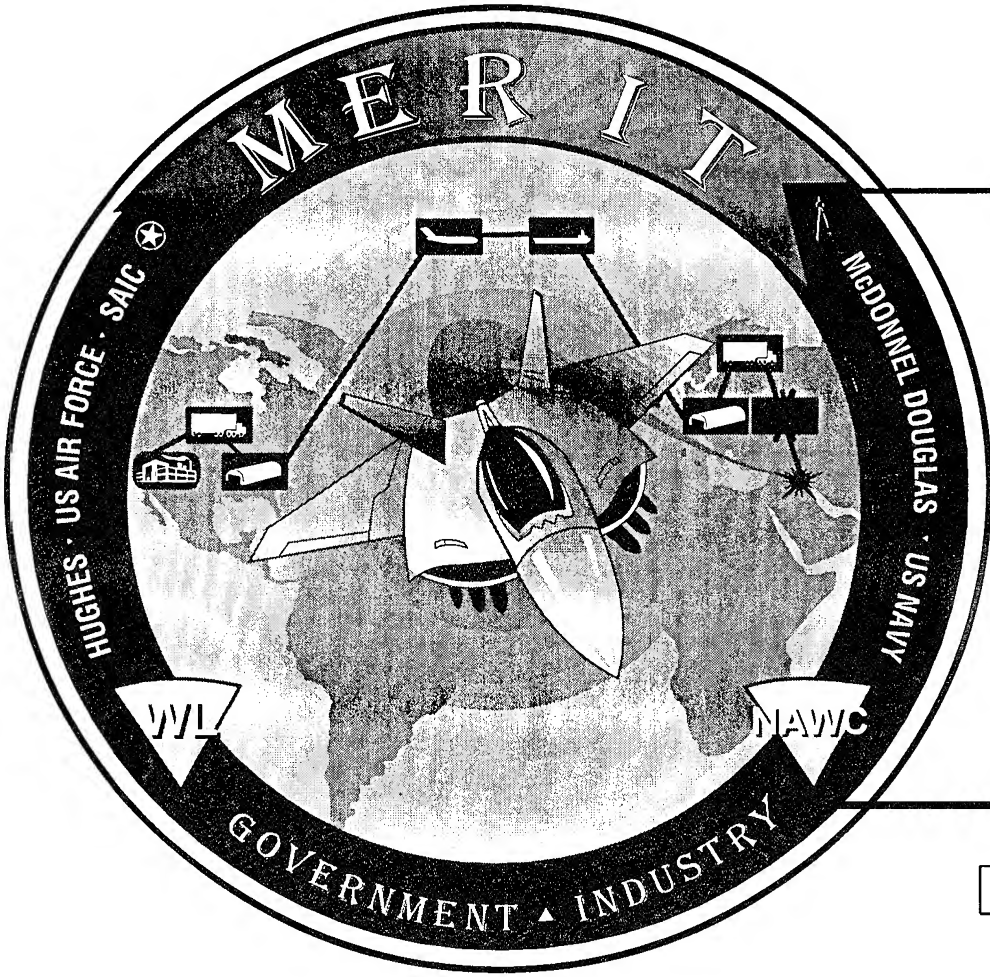
621

SUMMARY

INTELLIGENT APPLICATION OF DIAGNOSTICS CAN:

- √ HELP MITIGATE THE RISK OF USING G/COTS
- √ REDUCE PROPOSAL PRICE
- √ INCREASE WIN PROBABILITY
- √ <u>REDUCE YOUR COST!</u>

623



MISSION
ENVIRONMENTAL
REQUIREMENTS
INTEGRATION
TECHNOLOGY

BILL VONGUNTEN
SAIC, DAYTON, OHIO
PHONE: (513) 258-1170
FAX (513) 253-9765

MERIT OBJECTIVES

REQUIREMENTS ANALYSIS

- Integrate operational and logistics profiles with knowledge-based engineering to predict realistic life cycle environments during concept exploration

DECISION SUPPORT

- Quantify impact of logistics plan and weapon employment alternatives on engineering requirements
- Quantify impact of change in engineering requirements on operational capabilities (i.e., availability)

BACKGROUND

MERIT development program a result of WL/FIV experiences in:

- Equipment testing
  - CERT
  - MIL-STD-810
- ECS design
- Tire and wheel durability
- Aircraft canopy design and durability
- Mechanical subsystems integrity

625

FOUNDATION OF MERIT

- Early understanding of system-level events, loads, and environments needed to create effective, efficient development and acquisition programs
- Requires proper consideration of all phases of system life (i.e., beyond mission usage)
- Leads to the Life Cycle Environmental Profile (LCEP) as the fundamental basis for all design, assessment and test requirements

ATTRIBUTES OF THE LIFE CYCLE ENVIRONMENTAL PROFILE

- Time-Phased Sequence of Life Cycle Events (i.e., Cradle-to-Grave Chronology)
  - Manufacture and Testing Sequence
  - Deployment Scenario
  - Maintenance Concept
  - Field Usage (including Mission Profile)
- Environment-Event Correlation
  - Duration and Frequency of Occurrence
  - Natural and Induced Effects/Loads
  - Coexisting Environments
- Environment Predictions
  - Magnitude Levels (i.e., temperatures, g-rms, q, etc.)
  - Character (i.e., spectrum, rates of change, composition, etc.)

PERFORMANCE-BASED SPECIFICATIONS

- Concept behind Perry Initiative: "How we expect a system to perform should define our requirements for its design"
- Focuses on system-level performance
  - mission objectives (what & how)
  - operational availability (when and where)
  - cost (per unit and life cycle)

PROBLEM RECOGNIZED

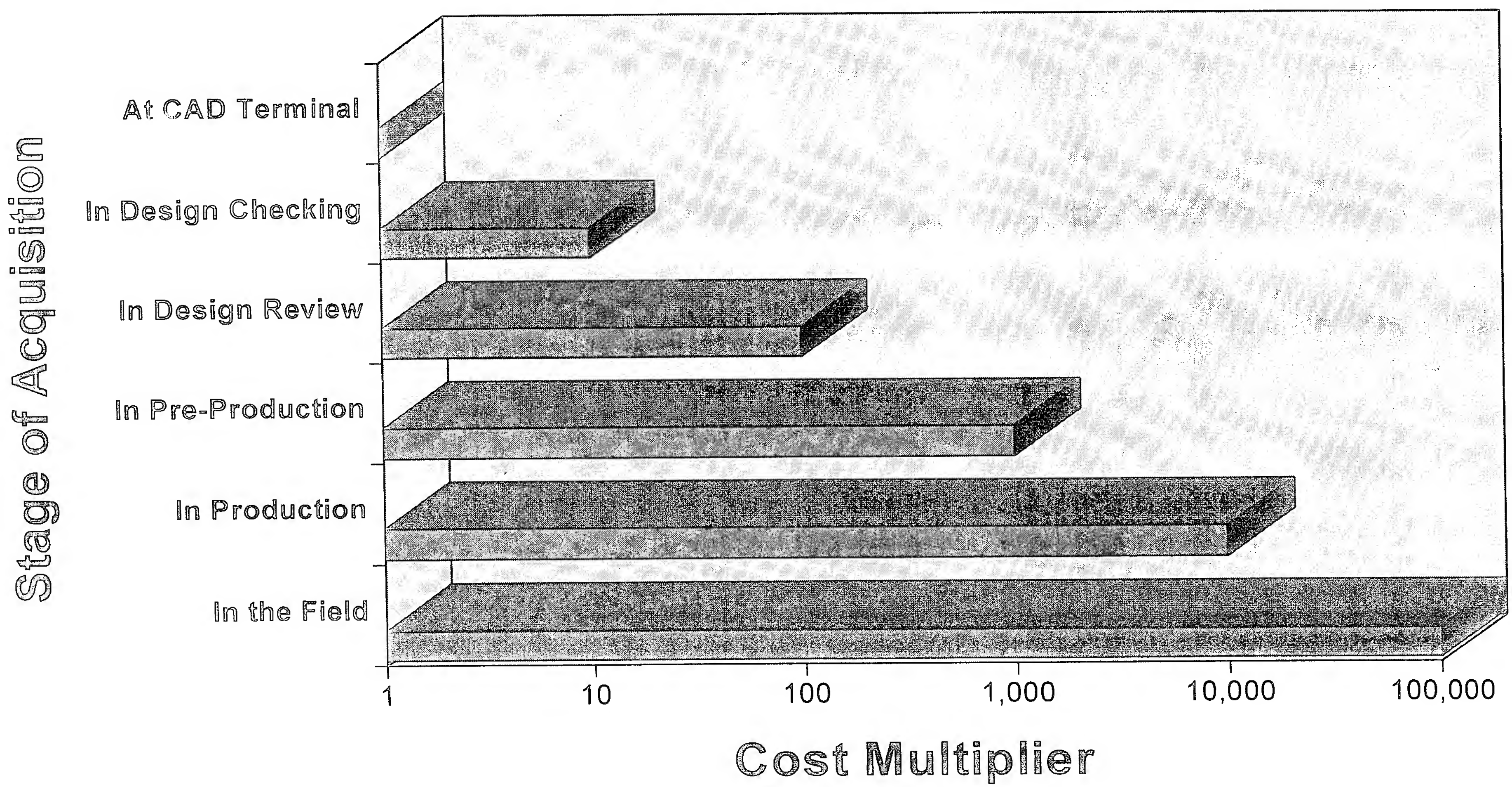
Realistic Life Cycle Environmental Profiles are not available to impact early design decisions in a Concurrent Engineering environment due to:

- **Manual process conflicts with design time compression (not CAD compatible)**
- **Data inaccessible, conflicting, or undocumented (lack of credibility)**
- **Limited pool of experience/expertise (not enough people for all programs)**

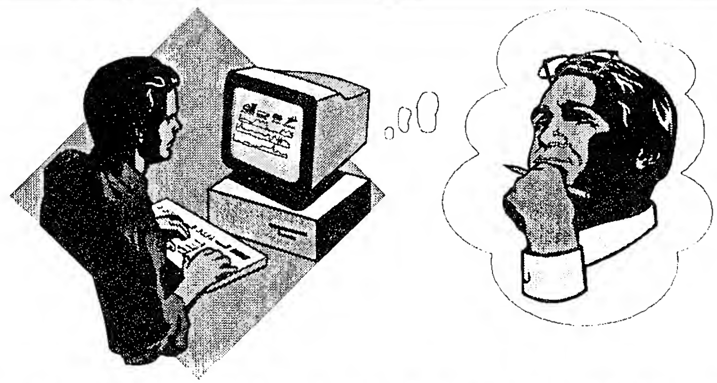
<u>Impact measured in terms of schedule and cost!!</u>

629

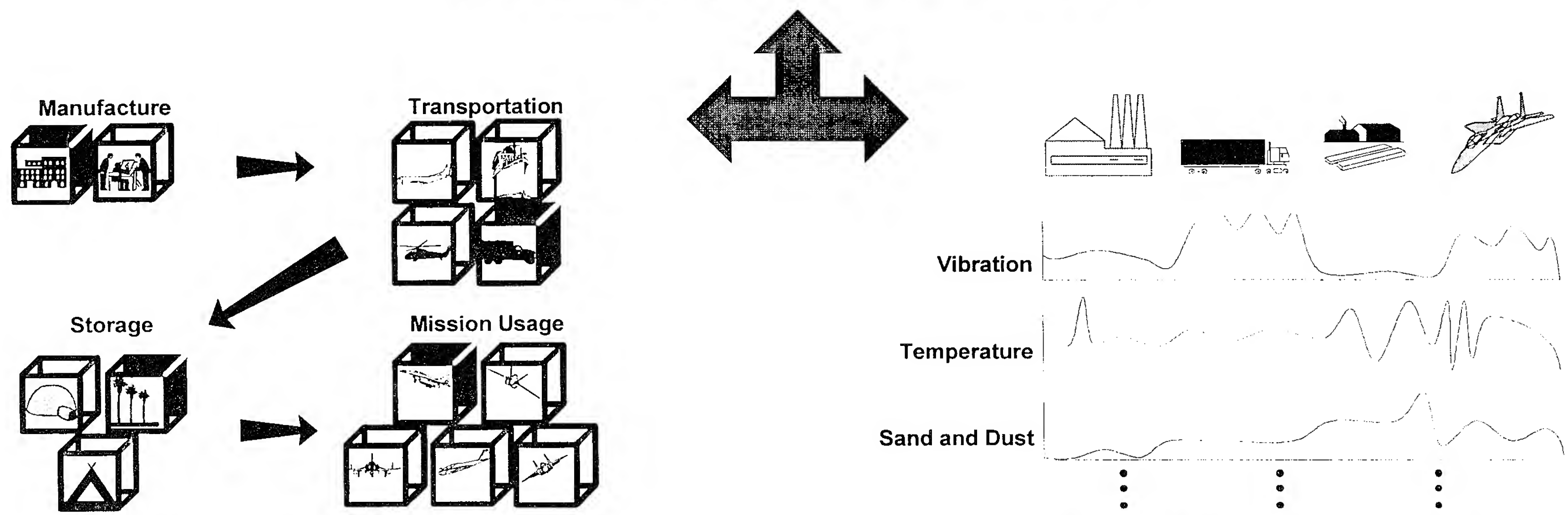
COST OF ENGINEERING CHANGES



MERIT CONCEPT: COMPUTER ASSISTED PROFILE GENERATION AND ENVIRONMENTS PREDICTION



EXPERT SYSTEM TECHNOLOGY



LIFE CYCLE DEFINITION

ENVIRONMENT PREDICTION

631

AIR FORCE MATERIEL COMMAND

ENABLING TECHNOLOGIES

- High Speed Personal Workstations (industry wide)
  - Fully interactive processing
  - Near real-time response
  - Sophisticated graphics rendering
- Software Frameworks (MS Windows)
  - Standardized data interface and protocol
  - Simplified data transfers
  - Reduced programming burden (i.e. less custom code)
- Improved Parametric Modeling Techniques (MERIT)
  - Automated linear and non-linear regression analysis
  - Linear and non-linear neural network development
  - Ability to mix data types via object oriented structures

MERIT SCOPE

- Define Environments and loads
  - Magnitude and duration
  - Establish boundary conditions
  - Provide basis for design/test requirements
- Emphasis of current effort: external stores on fighter/attack aircraft
  - Recognized harsh environment
  - Measured data available
  - Area of high activity
  - Most extensive life cycle profile

633

MERIT TEAM

- DoD
  - Wright Laboratory: FIV, FIB, DOWF
  - ASC: ENFS, ENAI, YAE
  - USAFETAC (Environmental Technical Applications Center)
  - NAWC-WD: Environmental Engineering Branch, China Lake
  - PL/GPAA: MIL-STD-210C OPR
- **McDonnell Douglas Aerospace - East**
- **Hughes Missile Systems Company**
- **Science Applications International Cooperation**

TECHNICAL APPROACH

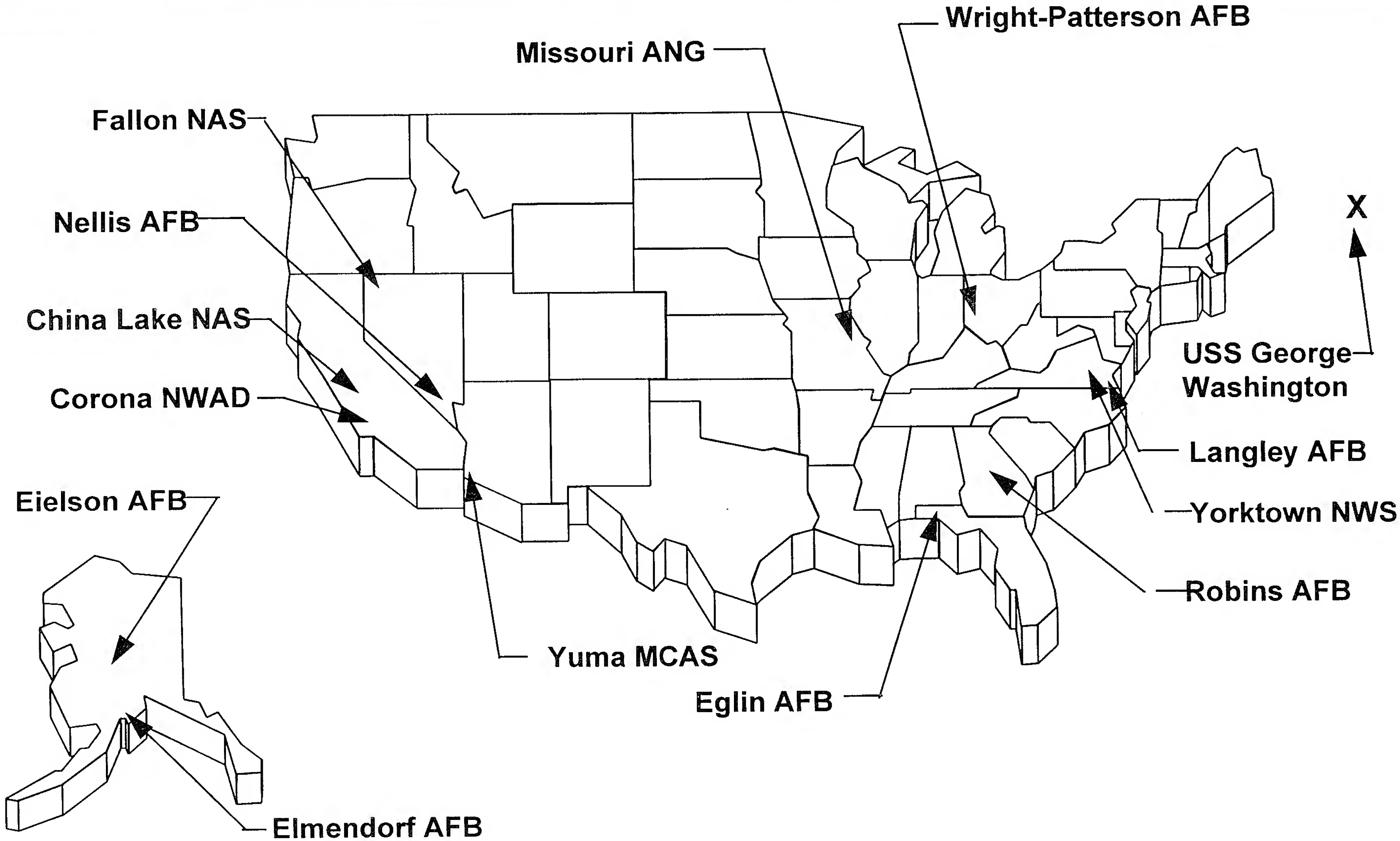
- Decompose life cycle into set of common, distinct activities to use as building blocks
- Develop environmental models which estimate the levels for each activity
- Use commercial computer-based, expert system techniques to link activities to models and assist user in life cycle profile creation
- Validate models and approach through operational usage at USAF and Navy sites

635

STORE/AIRCRAFT COMBINATIONS FOR LIFE CYCLE DEFINITION

Store \ Aircraft	AV-8	F-4 & F-4G	F-15 & F-15E	F-16	F/A-18
AMRAAM		O	O	O	O
Maverick	O	O	O	O	
Sidewinder	O	O	O	O	O
Sparrow		O	O	O	O
HARM		O			
JSOW					O
Harpoon				O	O
SLAM					O
NAVFLIR					O
LANTIRN			O	O	
AAAM			O	O	O
JDAM	O		O	O	
Tacit Rainbow			O	O	O

SITES OF LIFE CYCLE DATA AND KNOWLEDGE COLLECTION



AIR FORCE MATERIEL COMMAND

LIFE CYCLE ELEMENTS AND SETS

Handling	Storage	Transportation	Captive Carriage
Assembly At-Sea-Transfer Download Load Pack Short Distance Transfer Upload Unpack Upload	Covered Exposed Protected Ready	Air Rail Ship Truck	Air Refuel Climb Combat Climb Combat Descend Combat Level Flight Combat Turn Cruise Dash Descend Land Loiter Park Reconnaissance Search Takeoff Taxi

Weapon Flight	Testing	Servicing
Free Flight Launch Maneuver Flight Self-Generated Impact/Detonation	Bit Environmental Stress Screening/ Environmental Testing Functional Test Inspection Trouble Shooting	Clean Corrosion Prevention Disassembly Reassembly Repair

LIFE CYCLE ELEMENTS AND SETS

Handling	Storage	Transportation	Captive Carriage
Assembly At-Sea-Transfer Download Load Pack Short Distance Transfer Upload Unpack Upload	Covered Exposed Protected Ready	Air Rail Ship Truck	Air Refuel Climb Combat Climb Combat Descend Combat Level Flight Combat Turn Cruise Dash Descend Land Loiter Park Reconnaissance Search Takeoff Taxi

Weapon Flight	Testing	Servicing
Free Flight Launch Maneuver Flight Self-Generated Impact/Detonation	Bit Environmental Stress Screening/ Environmental Testing Functional Test Inspection Trouble Shooting	Clean Corrosion Prevention Disassembly Reassembly Repair

ENVIRONMENTS MODELED

- Pressure/Altitude
- High Temperature
- Low Temperature
- Temperature Change
- Solar Radiation
- Rain
- Snow
- Hail
- Ice
- Humidity
- Fungus
- Salt Fog/Salt Spray
- Corrosive Atmospheres
- Wind/Turbulence
- Sand/Dust/Aerosols
- Acceleration
- Vibration
- Acoustic Noise
- Shock
- Electromagnetic Interference (EMI)
- Electrostatic Discharge (ESD)
- Water Immersion
- Fluid Contamination
- Clouds
- Insects

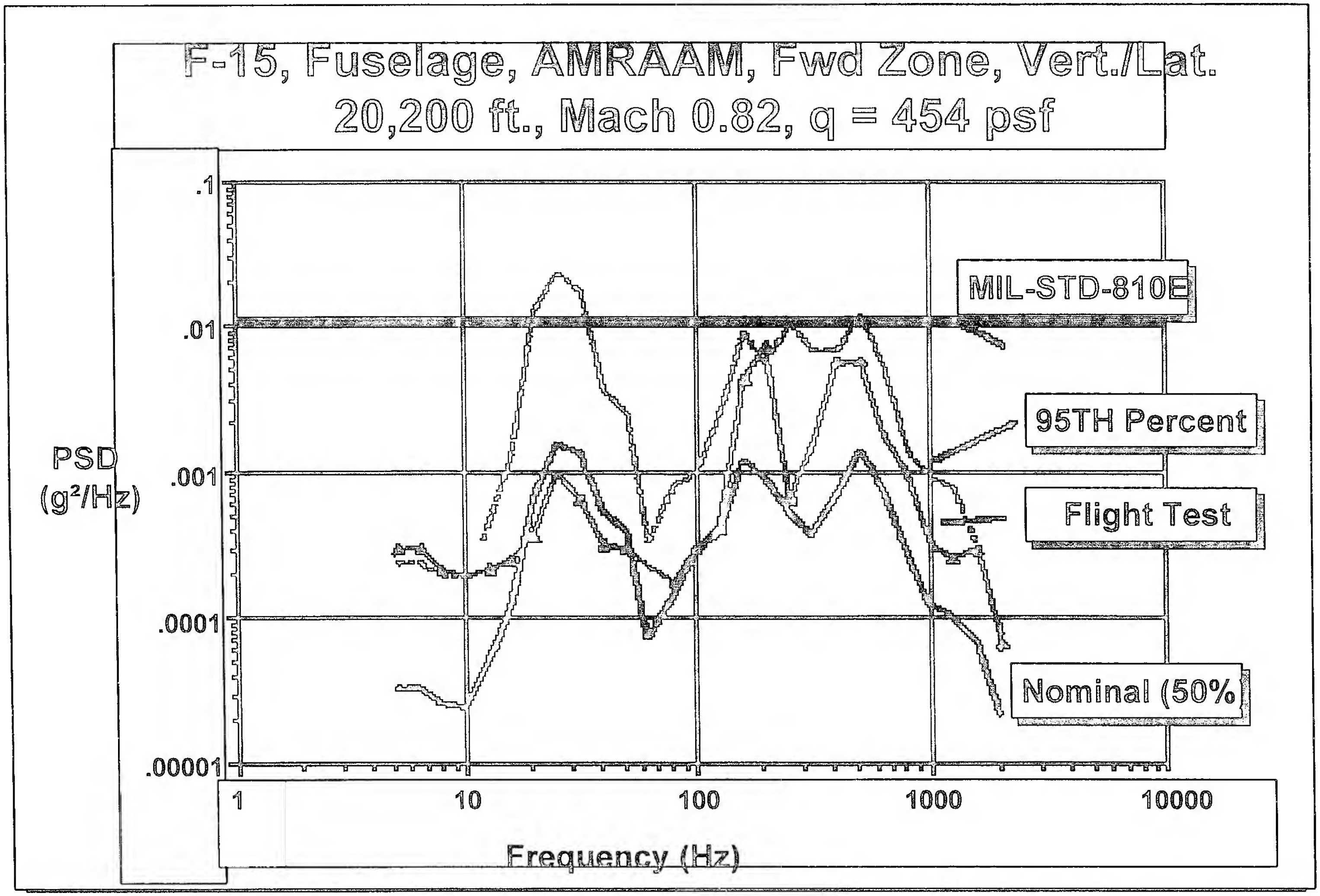
DYNAMIC MODELS EXAMPLE: FLIGHT VIBRATION

$$PSD = 10^\wedge\{ [B_O(f) + B_1(f)q^2] (W/V)^2 \} \; AF$$

- Parameter driven (i.e., q, geometry, mass, etc.)
- Scales each 1/3 octave band independently
- Calculates values for 3 zones and 2 directions
- Estimates nominal (50%) and 95% prediction values
- Accounts for location and airframe effects
- B_O and B_1 defined by log-linear regression of flight data
- W/V is weight to volume ratio

641

FLIGHT VIBRATION: AMRAAM ON F-15



CLIMATIC MODELS

- Based on USAFETAC data
- Using 17 year period of record with 3 hour observations
- Supplements MIL-STD-210
  - Climatic regions: 9 land, 4 ocean, 6 upper air (to 80K feet)
  - Monthly values with worst month by hour
  - Occurrence percentiles: 50, 80, 90, 95, 99, and Max
  - Freezing point included (32F, 0C), if in range
  - Duration data: Continuous period and total annual
- Allows determination of operational impacts

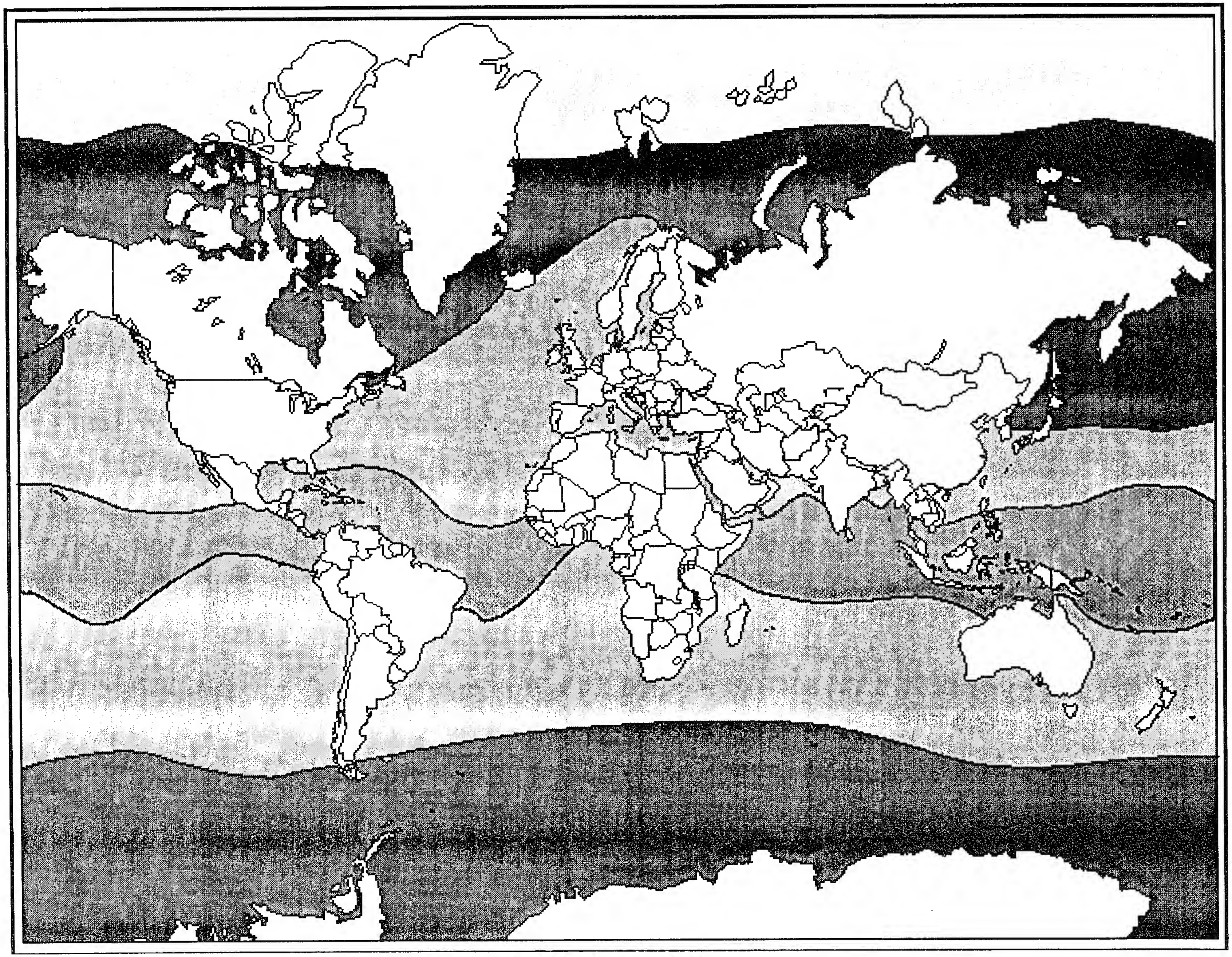
EXAMPLE: EIELSON AFB

Eielson AFB
Monthly Temperatures (°F)

MO	P0	P1	P5	P10	P20	P50	P80	P90	P95	P99	P100
1	-60	-53	-44	-38	-27	-5	10	18	27	40	47
2	-52	-45	-37	-31	-23	-5	13	23	30	39	50
3	-40	-29	-14	-8	-1	13	27	34	39	45	53
4	-21	-6	7	14	22	33	44	49	52	59	73
5	12	29	33	36	40	50	59	63	67	74	84
6	36	41	46	48	51	59	68	72	76	82	91
7	39	46	50	52	55	61	70	75	78	83	91
8	25	36	42	45	49	56	64	69	73	79	90
9	6	21	29	32	36	44	54	59	62	68	74
10	-35	-13	-1	6	14	26	35	40	45	52	62
11	-45	-34	-26	-20	-11	3	16	23	30	42	49
12	-53	-48	-39	-32	-21	-5	10	17	24	38	46
ALL	-60	-42	-26	-14	-1	31	55	63	68	77	91

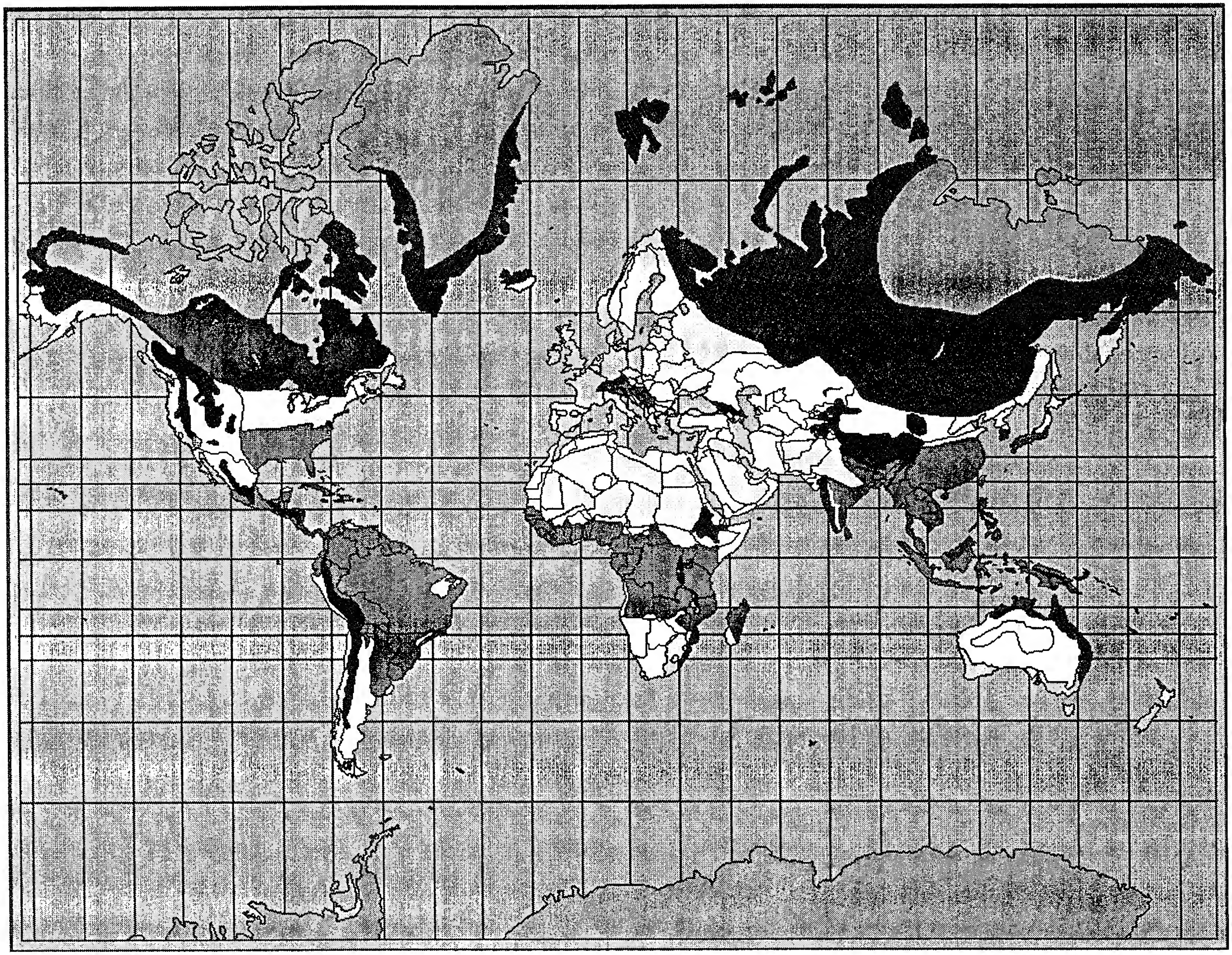
Eielson AFB
Temperature Duration

	CONTINUOUS HRS			ANNUAL HRS	
THOLD	MEDIAN	MEAN	MAX	AV	MAX
-60	5	5	6	0	6
-53	3	11	99	9	114
-44	6	12	126	68	378
-38	9	16	222	158	507
-27	9	18	378	414	966
-5	6	22	594	1484	2625
10	9	37	1533	2736	3645
18	9	49	1821	3361	5142
27	9	63	2292	4060	5283
32	9	65	3426	4499	6207
40	9	68	3882	5258	7131
47	9	66	5058	5954	7861



OCEAN REGIONS

☐ Arctic (permanent ice) ☐ Polar ($T \leq 10°C$, 1 ice free season) ☐ Midlatitude ☐ Tropical ($T \geq 25°C$)

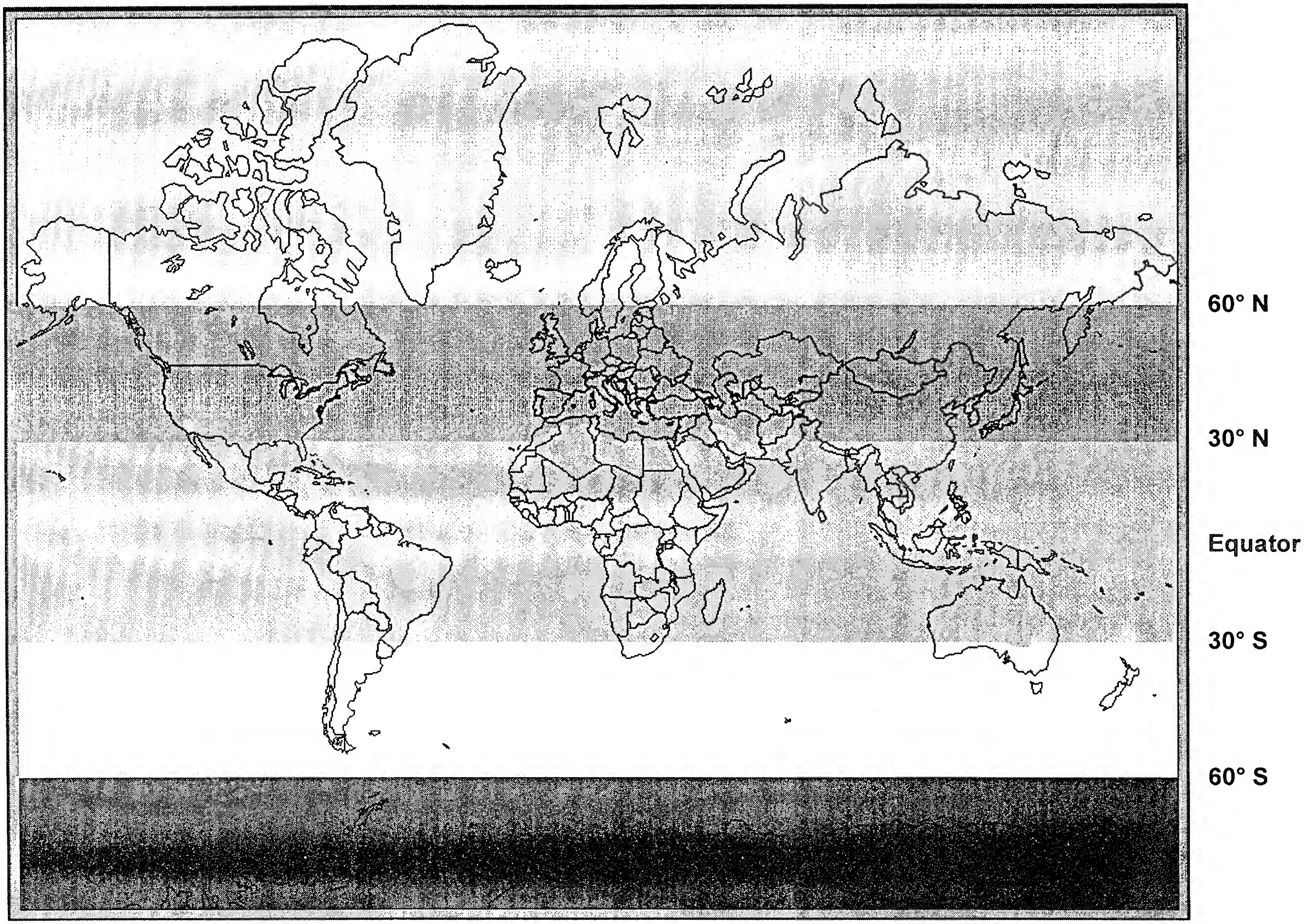


LAND REGIONS

☐ Severe Cold	☐ Marine	☐ Tropical
☐ Cold	☐ Dry	☐ Hot
☐ Continental	☐ Subtropical	☐ Highlands

645

647



UPPER AIR REGIONS

- ☐ Polar-North
- ☐ Midlatitude-North
- ☐ Desert
- ☐ Tropical
- ☐ Midlatitude-South
- ☐ Polar-South

ADDITIONAL FEATURES

- Tracks duration and frequency of occurrence for each life cycle activity and environment
- Provides a variety of options for life cycle and environmental data reports generation (data slicing)
- Based on commercial computer capabilities and familiar MS Windows interface
- Life cycle development process, as well as majority of elements and models, are equipment independent
- Provides near real-time access to accumulated corporate expert knowledge and lessons learned
- Provides a pedigree through on-line access to references, sources, and model details
- Supplements and expands upon MIL-STD-810F, MIL-STD-210C, AR 70-38, and the performance based specifications approach (re: Perry policy letter)

MERIT VALIDATION

- **MERIT involves use of new and innovative modeling approaches**
- **Requires extensive validation to insure predictions are within engineering tolerances which depend on:**
  - Stage of design
  - Criticality of environment and/or equipment
  - Magnitude of natural variances
  - Current state-of-the-art
- **Engineering validation being accomplished at two levels:**
  - Individual models
  - Integrated system

679

MODEL VALIDATION

- In development
  - Stable within defined range
  - Reproduce parent data set
  - Represents independent data set
  - Consistent with other methods
  - Documented history and references (available on-line)
- Pre-system testing
  - Distribute to selected team Program Offices
    - Currently includes: F-15, F-18, A/V-8, Harpoon, SLAM, JDAM, AMRAAM
    - Compare predictions to on-going flight tests and unreleased data
    - Evaluate using "engineering intuition"
  - Technical papers and presentations at technical meetings
    - Institute of Environmental Sciences (IES)
    - Shock and Vibration Symposium
    - Reliability and Maintainability Symposium

SYSTEM VALIDATION:
Beta Testing

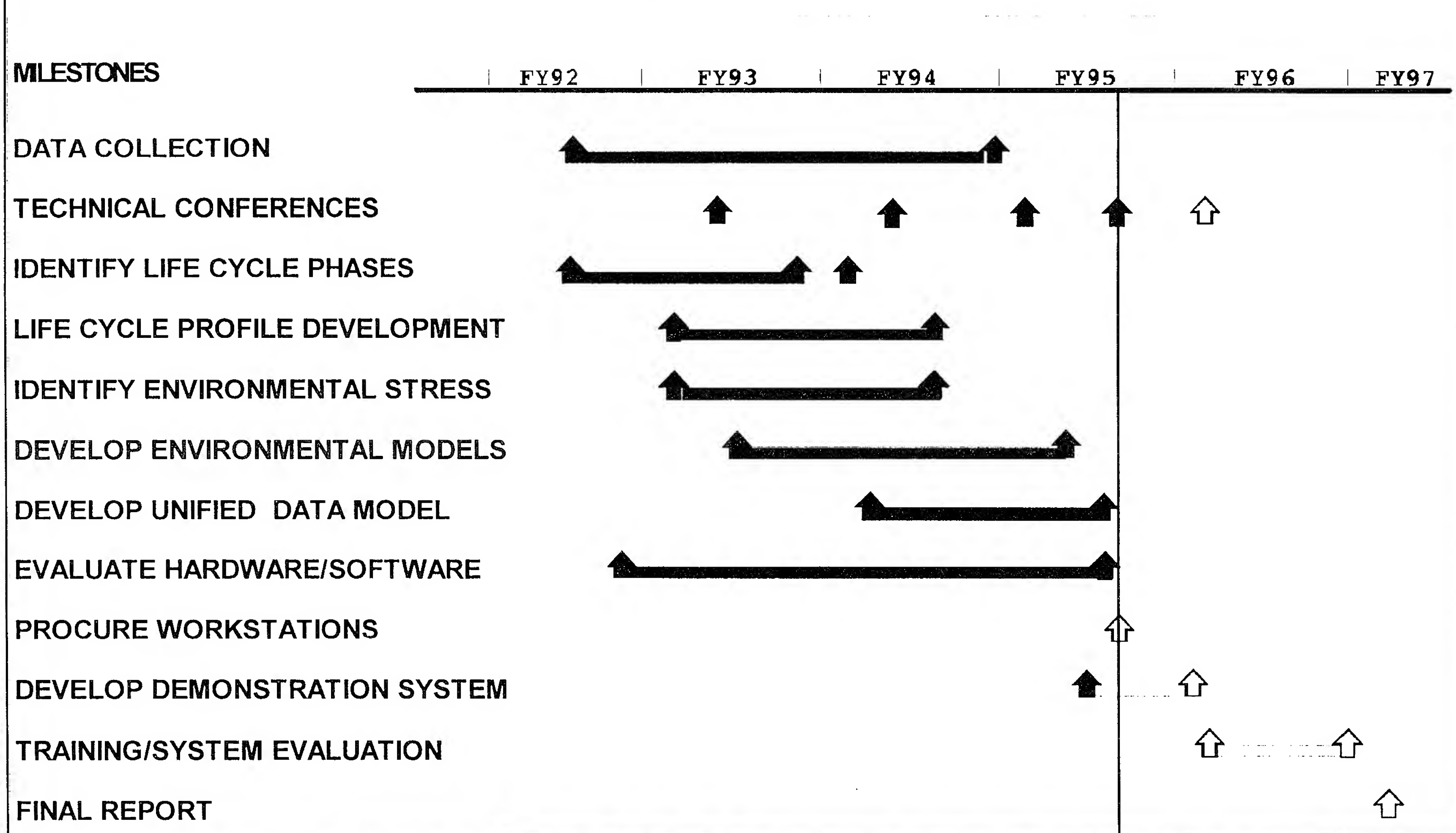
- Provide system and training to 6 DoD engineering sites:
  - WL/FIV
  - ASC/ENFS
  - WR-ALC/LKGE
  - USAFETAC
  - NAWCWPNS
  - Eglin AFB? (perhaps ASC/YAE or WL/MN)
- Evaluate under typical engineering usage for 8 months
- Require periodic feedback on system performance and accuracy
- Provide contractor supported maintenance and improvements prior to formal release

OPERATION, MAINTENANCE AND SUPPORT (OM&S)

- Issues
  - Distribution and customer support
  - Access to and quality control of models
  - Technical viability and currency
  - Inclusion of other systems
  - Expansion to other engineering domains
- **Reality: WL has a history of being a good system developer, bad system maintainer**
- SAIC developing MERIT OM&S concept alternatives with discussion of pros and cons and recommendations

MILESTONE SCHEDULE

PROGRAM TITLE: MISSION ENVIRONMENTAL REQUIREMENTS INTEGRATION TECHNOLOGY (MERIT)



653

MERIT FUNDING PROFILE ($K)

FY92	FY93	FY94	FY95	FY96	FY97	TOTAL
444	2341	2648	2269	1321	119	9142

AIR FORCE MATERIEL COMMAND

FUTURE OF MERIT

654

Increasing need for usage-based loads and environments prediction capability to support effective, efficient acquisition of military equipment. Areas that MERIT can impact include:

- Service Life Extension/Aging Systems
- Use of Non-Developmental Items (NDI)/COTS
- Performance Modeling and Simulation
- MIL-STD-210 and 810 updates and computerization
- Flight test data reduction, storage, and recall

655

CONCLUSIONS

- Better Requirements = Improved Performance
- Loads and environments definition must become partner in Concurrent Engineering process (i.e., CAD compatible formats)
- Requires automated/rapid access to existing data and knowledge
- MERIT addresses needs without constraining the engineer
- Technology can be used to assist other functions in the equipment acquisition process to improve system cost, reliability and performance